National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

University of Alberta

Extract Answers in Circumscription and Logic Programs: A One-Staged Proof-Theoretic Framework

by

Chenghui Wang

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Department of Computing Science

Edmonton, Alberta
Spring 1996

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-10650-0

Canada

University of Alberta

Library Release Form

Name of Author: Chenghui Wang

Title of Thesis: Extract Answers in Circumscription and Logic Programs: A One-Staged Proof-Theoretic Framework

Degree: Doctor of Philosophy

Year this Degree Granted: 1996

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Chenghui Wang

Chenghui Wang
3 - 75 Woodridge Crescent
Nepean, Ontario
Canada K2B 7T2

Date: April 16, 1996

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Extract Answers in Circumscription and Logic Programs: A One-Staged Proof-Theoretic Framework** submitted by Chenghui Wang in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dr. L. Yuan

Dr. J.H. You

Dr. P. van Beek

Dr. L. Marcoux

Dr. R. Goebel

Dr. L.V.S. Lakshmanan

Date: March 15, 1996

Abstract

McCarthy's Circumscription [31, 30] is one of the most powerful formalisms aimed at characterizing the nonmonotonic aspect of commonsense reasoning. Many of the other nonmonotonic reasoning formalisms, as well as a class of PROLOG programs and deductive databases, can be expressed in circumscriptive formulas.

Circumscription is difficult to compute, because of the second-order quantifier involved in its definition. The existing proof procedures for circumscription [18, 36] require that the given set of clauses be a complete theory, because their progress depends on the subresolutions to terminate. Due to the semi-decidability of first-order logic, these proof procedures basically deal with only finite ground theories and give only YES or NO answers.

In this Ph.D. dissertation we redefine circumscription procedurally. We then develop *CP-resolution* (construct P) which is orthogonal to the resolution tree of the procedural definition. CP-resolution constructs the definition of a single negative literal under circumscription. It overcomes the inherent limitations of the other nonmonotonic proof procedures such as *floundering*, unnecessary incompleteness, inability to extract answers in negation, and inability to extract partial answers when the completion condition is not met. It is proved to be complete under the *weakly bounded term size property*, which is known to be the weakest completion condition for a nonmonotonic proof procedure, and it produces partial answers when the completion condition is not met. CP-resolution is a proof-theoretic framework in the sense that proof procedures of other semantics for logic programs and deductive databases can be developed, which retain all the advantages of CP-resolution.

Based on CP-resolution, we develop *CIR-resolution* (compute CIRcumscription), which can extract answers in circumscription. CIR-resolution is complete if the given set of clauses possesses the weakly bounded term size property. It can extract partial answers even when the completion condition is not met.

Based on Yuan's work [48], where two of the best known semantics for logic programs with negation, *stable* and *well-founded* semantics, are redefined in terms of circumscription of autoepistemic theories, we further develop *ST-resolution* (compute

STable semantics) and *WF-resolution* (compute Well-Founded semantics) which extract answers in logic programs under the stable and well-founded semantics. Both ST-resolution and WF-resolution are complete under the condition of the weakly bounded term size property. They can extract partial answers even when the completion condition is not met.

Contents

1 Introduction **1**

1.1 Circumscription 2

1.2 Nonmonotonic Proof Procedures 3

1.2.1 Limitations of the Existing Proof Procedures 4

1.2.2 Floundering 4

1.2.3 Unnecessary Incompleteness 5

1.2.4 Lack of Answer Substitutions 6

1.2.5 Partial Answers 7

1.3 Sources of the above Limitations 7

1.3.1 Negation as Failure 8

1.3.2 Ordering Dependent Resolutions 8

1.3.3 Multi-staged Resolution 8

1.4 The Key Ideas of the Dissertation Research 9

1.4.1 Constructive Negation 9

1.4.2 Blocking Formulas 10

1.4.3 Circumscription-Based Predicate Completion 10

1.5 The Major Contributions of the Dissertation 11

1.5.1 CP-resolution 11

1.5.2 Extracting Answers in Circumscription 11

1.5.3 Proof Procedures for Well-founded and Stable Semantics of Logic Programming 12

1.6 Comparison and Future Work 12

1.7 Organization of the Document 13

2 Preliminary **15**

2.1 Syntax of the Logic Theories 15

2.2 Interpretations and Models 17

2.2.1 Free Universes and Free Interpretations 20

2.3 Deletion Rules 21

2.4 Skolem Expansions 21
2.5 Unification 23
2.6 Clark's Equational Theory (CET) 24
2.7 Clark's Predicate Completion 26
2.8 Constructive Negation Answers 27
2.9 Circumscription 28
2.10 Assumptions 30

3 CP-resolution: A Proof-Theoretic Framework **31**
3.1 Local Skolemization and Blocking Formulas 31
3.2 Rules of Inference 35
3.3 CP-resolution 39
3.4 MIME-resolution, MIME-tree and the Weakly Bounded Term Size Property 42
3.5 Orthogonality of CP-resolution and MIME-resolution 46
3.6 Procedural Definition of Circumscription 48
3.7 Property Analysis 54

4 Extracting Answers in Circumscription **58**
4.1 CIR-resolution: A Proof Procedure to Extract Answers in Circumscription 58
4.2 Soundness and Completeness 59
4.3 Extended Results 64
4.3.1 CIR-resolution for Prioritized Circumscription 64
4.3.2 Answer Generic Query with CIR-resolution 65
4.4 Comparison with Other Works 67
4.4.1 Przymusinski's Query Evaluation Algorithm and MILO-Resolution 68
4.4.2 Weakness with Przymusinski's Query Evaluation Algorithm . 71

5 Computing Logic Programs **76**
5.1 Two Semantics: Stable and Well-founded 76
5.2 ST-resolution: A Proof Procedure for Stable Semantics 79
5.3 WF-resolution: A Proof Procedure for Well-founded Semantics 83

6 Conclusion **90**

Bibliography **92**

A Justification of the Way MIME-resolution is Defined **96**
A.1 Why Introduce Ground WME-resolution? 96
A.2 Why in General Form? 97

B Are Variant Chains Really Needed? **99**
B.1 Variant Chain Issue with CIR-resolution 99
B.2 Variant Chain Issue with ST-resolution 101
B.3 Variant Chain Issue with WF-resolution 101

C Complicated Proofs **103**

Chapter 1

Introduction

Even though first-order predicate cálculus has been widely used in database and artificial intelligence systems as a language for expressing declarative knowledge, it has major limitations. The three most significant ones [17] are:

1. A knowledge base could never store all the information that we want about the world, especially the overwhelming amount of negative information. Any general rule that we may care to frame is subject to unlimited number of *exceptions* and *qualifications*. The first-order language cannot capture such incomplete information and unlimited exceptions.

2. The inference rules of first-order logic are sound and thus the deduction from a base set of beliefs never produces new knowledge about the world. Consequently, we need to add new formulas to our base of beliefs that say new (or revised) things about the world. Ordinary logic, however, gives us no hints about how to do that. We need reasoning methods which will anticipate the possibility of later revision of the knowledge base.

3. Ordinary logic languages are adequate for expressing only those statements that we are willing to take as being either wholly true or wholly false, despite the fact that we often have information about a situation that is known to be *uncertain*.

Many techniques have been proposed to confront and overcome some of these limitations. An important technique involves the use of *nonsound* inference rules of various kinds. That is, from a knowledge base Π, we allow certain inferences that do not logically follow from Π. These inferences often depend globally on all of the sentences in Π rather than a small subset. With such inference rules, if another sentence is added to Π, an inference may have to be retracted. For this reason, such

inference rules are called *nonmonotonic*. On the other hand, ordinary logic inference rules are monotonic, because the set of theorems derivable from the premises is not reduced by expanding the premises.

There are a number of ways to formalize reasoning with nonmonotonic effects. Among the others, Reiter's *Closed World Assumption* [43] and *Default Logic* [41], Moore's *Autoepistemic Logic* [34], and McCarthy's *Circumscription* turn out to be the best known nonmonotonic reasoning formalisms.

Circumscription is a nonmonotonic reasoning formalism proposed by John McCarthy [31, 30] based on the notion of truth in all minimal models of a first-order theory. It is recognized as one of the most powerful formalisms proposed for nonmonotonic reasoning due to its expressive power. In fact, it has been shown that almost all prominent nonmonotonic reasoning semantics, including the extensional semantics of the Reiter's default logic, the stable semantics of Moore's autoepistemic logic, the closed world assumption, and various declarative semantics of logic programs with negation, can be expressed in terms of circumscription of autoepistemic theories [47, 48].

Logic Programming is of crucial and growing significance to the field of artificial intelligence and therefore to the field of computer science. It introduces the important concept of declarative, as opposed to procedural, programming based on mathematical logic. Yet logic programs do not use logical negation, but rely on nonmonotonic operators, such as *negation as failure* [8, 27].

A sophisticated proof procedure for circumscription would thus be of great significance, for it would not only compute circumscription, but also compute most prominent nonmonotonic reasoning semantics as well as various declarative semantics of logic programs and deductive databases.

In this introduction, we first examine the existing nonmonotonic proof procedures and their limitations. We explain the key ideas used in this dissertation. The main contributions are then presented, at the center of which is the proof-theoretic framework, *CP-resolution*, upon which the resolutions for circumscription, *stable* and *well-founded* semantics of logic programming are developed. We compare CP-resolution with other nonmonotonic operators, to show how it overcomes the inherent limitations of other nonmonotonic proof procedures.

1.1 Circumscription

In this section we give an informal description of circumscription .

A promising way of achieving nonmonotonicity is to treat the sentences that are true in all the preferred models of a logical theory as theorems. These preferred

models possess the property of minimality of some kind. Circumscription is such a nonmonotonic reasoning formalism based on the notion of truth in all P-minimal models of a first-order theory, as explained below.

Let $\Pi(Q, P, Z)$ be a first-order theory such that Q, P and Z are disjoint classes of predicate symbols. Intuitively, the predicates in P are *abnormality* predicates we wish to minimize; the predicates in Q represent facts; and the predicates in Z are those that we don't care too much about and therefore, in order to give more room for P to be minimized, are allowed to vary.

Two models M_1 and M_2 of the theory $\Pi(Q, P, Z)$ can compare with each other only if they have the same interpretation of the predicates from Q. M_1 is said to be P-smaller than M_2, denoted as $M_1 \leq M_2$, if both models assign the same relations to the predicates in Q, but the relations assigned to the predicates in P by M_1 are subsets of those assigned by M_2. We don't care how the models assign the relations to the predicates from Z. A model M_1 is a P-minimal model of Π if for any other model M_2 of Π, $M_2 \leq M_1$ implies $M_1 \leq M_2$.

A model of the theory Π is a model of the circumscription of Π on P with Z varying, denoted as $CIR(\Pi; P; Z)$, if and only if it is a P-minimal model of Π. A sentence is implied by $CIR(\Pi; P; Z)$ if and only if it is true in all the P-minimal models of Π.

1.2 Nonmonotonic Proof Procedures

Assume Π is a logic theory. We use $SEM(\Pi)$ to denote the (possibly infinite) set of all sentences derivable under a given semantics of Π.

Given semantics $SEM(\Pi)$ and query $p(\boldsymbol{x})$ (which can also be written in the oriented form, called a goal, $\leftarrow p(\boldsymbol{x})$), where $\boldsymbol{x}$ stands for a tuple of free variables, a proof procedure is a computational procedure that determines if $SEM(\Pi) \models p(\boldsymbol{\alpha})$ for some $\boldsymbol{\alpha}$ in the underlying universe. A proof procedure is

1. *sound* if every answer returned from the procedure is correct with respect to the query under the semantic;

2. *complete* if the procedure will return all correct answers with respect to the query under the semantics.

Obviously, any proof procedure should be sound with regard to the semantics, though it may not be complete.

Here are some widely known proof procedures for nonmonotonic reasoning.

1. *SLDNF-resolution* (SLD-resolution with Negation as Failure) [27];

2. *SLD-CNF-resolution* (SLDNF-resolution with Constructive Negation) [5]; and
3. *MILO-resolution* (MInimal model Linear Ordered resolution) [36].

1.2.1 Limitations of the Existing Proof Procedures

Due to the undecidability of first-order logic, almost all nonmonotonic proof procedures are inherently incomplete in that not all correct answers can be derived under a given proof procedure.

We first discuss the main limitations with the existing nonmonotonic proof procedures such as *floundering*, *unnecessary incompleteness*, lack of *answer substitutions*, and inability to *extract partial answers* when the completion condition is not satisfied.

1.2.2 Floundering

A query *flounders* if at some point in the computation, the goal contains only nonground negative literals. Most nonmonotonic proof procedures have to abandon floundered queries simply because further pursuit of a floundered query might lead to an unsound answer, as shown in the following example.

Example 1.1 ([27] page 93) Let $\mathcal{P}$ contain

$$p_1 \leftarrow \neg p_2(x); \quad p_2(a) \leftarrow$$

and the given goal be $\leftarrow p_1$. Under the SLDNF-resolution, the derivation is:

G_0: $\leftarrow p_1$;
G_1: $\leftarrow \neg p_2(x)$;

The query is floundered at this point. Further pursuit of the answer with a floundered query could lead to an unsound result.

G_0': $\leftarrow p_2(x)$;
G_1': $\leftarrow \square$.

The subgoal $\neg p_2(x)$ fails because there is a refutation of $\leftarrow p_2(x)$ in which x is bound to a. However, the answer is not correct because $\neg p_1$ is not a logical consequence of the negation as failure semantics of $\mathcal{P}$. $\square$

Not all floundered queries lead to unsound answers, however, it is undecidable to determine whether it is the case [44]. Discarding of floundered queries, though avoids unsound answers, might cause unnecessary incompleteness, because possible answers are discarded along with it.

1.2.3 Unnecessary Incompleteness

In this section we focus on the unnecessary incompleteness caused by a fixed selection function.

Consider the SLDNF resolution with the select function that always selects the rightmost subgoal whenever possible, as presented in [27].

Example 1.2 ([27], page 56) Consider the program $\mathcal{P}$

C_1: $p(x,z) \leftarrow p(y,z), q(x,y);$
C_2: $p(x,x) \leftarrow$
C_3: $q(a,b) \leftarrow$

and the goal $\leftarrow p(x,b)$. The set of all possible derivations are:

Derivation 1:

	Goals	Bindings	Input Clause
G_0:	$\leftarrow p(x,b);$		C_2
G_1:	$\leftarrow \square;$	$\{x/b\}$	

Derivation 2:

	Goals	Bindings	Input Clause
G_0:	$\leftarrow p(x,b);$		C_1
G_1:	$\leftarrow p(y,b), q(x,y);$		C_3
G_2:	$\leftarrow p(b,b);$		C_2
G_3:	$\leftarrow \square;$	$\{x/a\}$	

Derivation 3 can be carried on from G_2 of Derivation 2:

	Goals	Bindings	Input Clause
G_2:	$\leftarrow p(u,b), q(b,u);$		C_1
	failure		

All the derivations are terminated and therefore, the answer is complete. Since a binding represents the substitution of a variable in the query with a term, the answer can be expressed in the form of an equality formula:

$$\forall x\,(e(x,a) \vee e(x,b)).$$

If C_1 is reformulated as $p(x,z) \leftarrow q(x,y), p(y,z)$, which effectively change the selection function, then there would be an infinite branch in the tree:

G_0: $\leftarrow p(x,b);$
G_1: $\leftarrow q(x,y), p(y,b);$
G_2: $\leftarrow q(x,y), q(y,u), p(u,b);$

G_3: $\leftarrow q(x,y), q(y,u), q(u,v), p(v,b);$

.....

infinite

If the query is $\neg p(x,b)$, we would never have any answer, because the derivation with goal $\leftarrow p(x,b)$ never terminates. Note, here we assume that the floundering problem can be correctly overcome. The intuitive answer to the query $\neg p(x,b)$ should be the equality formula:

$$\forall x\,(\neg e(x,a) \wedge \neg e(x,b)).$$

□

1.2.4 Lack of Answer Substitutions

In developing query answering systems for nonmonotonic reasoning and logic programming, we are not merely interested in whether a query $p(\boldsymbol{t})$ can be derived, but also in which instances of $p(\boldsymbol{t})$ are derivable, commonly referred to as *answer substitutions*. It is generally agreed that the success of logic programming languages is largely due to their ability to produce proper answer substitutions [33].

For logic programs without negation and many other monotonic query answering systems, proper answer substitutions are not very difficult to achieve. The lack of proper answer substitutions, however, is one of the weak points for many nonmonotonic reasoning systems with negation.

For example, for a positive subgoal $p(\boldsymbol{x})$, the binding substitution θ obtained through the refutation of the proof procedure means that SEM $\models \forall p(\boldsymbol{x})\theta$. This can be written equivalently as SEM$\cup$CET $\models \forall(e(\boldsymbol{x}, \boldsymbol{x}\theta) \rightarrow p(\boldsymbol{x}))$, where e is an equality, and CET is *Clark's equational theories* [8, 27] which interpret the equality formula. Thus the binding substitution θ, or its corresponding equality formula $\forall x\, e(\boldsymbol{x}, \boldsymbol{x}\theta)$, is the *computed answer* for the query $p(\boldsymbol{x})$.

According to Chan [5], the answer to the query $\neg p(\boldsymbol{x})$ should be $\forall\,(\neg E)$, if the answer to the query $p(\boldsymbol{x})$ is $\forall\,(E)$, where E is an equality formula. To achieve this, the following requirements have to be satisfied:

1. The answer E to the query $p(\boldsymbol{x})$ must be in a form which can be negated;
2. The answer E is complete.

A binding substitution θ is certainly not in a form to be negated. Moreover, a binding substitution θ, where $\theta = \{\boldsymbol{x}/\boldsymbol{t}\}$, obtained through a refutation of a proof procedure means that there exists some instance, not necessarily every instance,

of $p(\boldsymbol{x})$, which is TRUE with respect to the semantics. Interpreting the binding substitution as TRUE, and thus its negation as FALSE, would lead to an unsound answer to the query $\neg p(\boldsymbol{x})$.

Obviously, Requirement (2) is much more difficult to be satisfied than Requirement (1). Because of such a difficulty many nonmonotonic proof procedures, represented by SLDNF-resolution [27], simply discard floundered queries. Others like SLD-CNF resolution [5] and SLSC-resolution (Linear resolution with Selection function for Stratified program enhanced with Constructive negation) [37] are able to extract answers from a negative subgoal only if Requirement (2) is satisfied.

1.2.5 Partial Answers

Soundness and the condition to achieve completeness are two of the most important criteria to evaluate a nonmonotonic proof procedure. However, we argue that the ability to derive partial answers is a criterion almost as important. Consider two nonmonotonic proof procedures that have the same completion condition, but one is able to derive partial answers when the completion condition is not met, while the other is not. If all others are equal, we do prefer the former to the latter.

The nonmonotonic proof procedures based on the negation as failure operator can give all the answers when the completion condition is met, but NONE otherwise, as shown in the following example.

Example 1.3 Let $\mathcal{P}$ contain $\{p(x) \leftarrow q(x), p(f(x));\ \neg q(a)\}$, and the query be $\neg p(x)$. Assume the selection function selects the rightmost subgoal whenever possible. We need first to compute $p(x)$ by means of the negation as failure rule.

The derivation is:

G_0: $\leftarrow p(x)$;
G_1: $\leftarrow q(x), p(f(x))$;
G_2: $\leftarrow q(x), q(f(x)), p(ff(x))$;
...

The derivation never terminates, though the equality formula $\forall x\, e(x, a)$ is an answer to the query $\neg p(x)$. □

1.3 Sources of the above Limitations

The main cause of the limitations discussed in Section 1.2.1 is the *use of the negation as failure operator.* The other reasons are *ordering dependent resolutions* and *multi-staged resolutions.* We discuss these sources in this section.

1.3.1 Negation as Failure

The notion *negation as failure* states that if a ground formula is proved not provable, then infer its negation [8, 27, 15]. Negation as failure only applies to a ground subgoal, and an attempt to use it in a general case often leads to unpredictable consequences.

One notable example, as demonstrated earlier, is that the evaluation of a floundered query using negation as failure might lead to an unsound answer. This is because pursuing a floundered query with negation as failure changes an existential query into a universal one. Consider a goal $\leftarrow \neg p(x)$ which states that the goal will succeed if $\exists x \neg p(x)$ can be proved. The negation as failure, however, considers the goal $\leftarrow p(x)$ and will fail the goal $\leftarrow \neg p(x)$ if $\exists x p(x)$ can be proved, even though both $\exists x p(x)$ and $\exists x \neg p(x)$ may be proved.

The use of the negation as failure operator is responsible for not just the floundering but also unnecessary incompleteness. This is because negation as failure works only for complete theories, where the inference engine is guaranteed to terminate. That is, only when all the derivations are terminated can we say that the complete answers are computed.

1.3.2 Ordering Dependent Resolutions

In order to reduce the search space, most resolution procedures use a selection function to determine the ordering of subgoals to be evaluated, and the ordering is usually determined by the ordering of the literals in each input clause. A proper ordering ensures that the subgoal which will eventually fail is always selected ahead of the one which will develop into an infinite branch, and therefore, avoid unnecessary incompleteness. However, finding an proper ordering is clearly an undecidable problem.

On the other hand, as we saw from Section 1.2.3, ordering dependency of the nonmonotonic resolutions is responsible for the unnecessary incompleteness. To overcome this limitation, we need a nonmonotonic operator which is insensitive to the ordering among the literals in each clause.

1.3.3 Multi-staged Resolution

An i-staged resolution is recursively defined as follows:

1. A resolution is one-staged if its progress does not depend on any resolution tree to be complete;

2. A resolution is i-staged if at a certain point it has to stop and wait for some $(i-1)$-staged resolution tree to be complete.

ME-resolution (Model Elimination) [29], *SL-resolution* (Linear resolution with Selection function) [22], *OL-resolution* (Ordered Linear Resolution) [7] and *SLD-resolution* (SL-resolution for Definite clauses) are typical examples of one-staged resolutions due to their strict linearity.

Any proof procedure that is based on negation as failure is a *multi-staged* procedure, since whenever a negative subgoal is selected, the derivation requires the completion of the whole resolution tree of a positive literal goal. SLDNF-resolution [27], SLD-CNF resolution [5], SLS-resolution (Linear resolution with Selection function for Stratified program) [38], and SLSC-resolution (SLS-resolution with Constructive negation) [37] are all multi-staged resolutions.

However, the proof procedure which does not use negation as failure operator could also be multi-staged. For example, Przymusinski's *query evaluation algorithm* with respect to the semantics of circumscription [36] is 3-staged.

Multi-stagedness is the main reason why the proof procedure can not extract partial answers. It features *nothing or all* query answering systems.

1.4 The Key Ideas of the Dissertation Research

We discuss briefly the key ideas used in this research.

1.4.1 Constructive Negation

The traditional answer in the form of a binding substitution is adequate to capture the meaning of a positive information such as "*all*". A computed binding substitution answer θ to the query $p(\boldsymbol{x})$ means that $SEM \models \forall p(\boldsymbol{x})\theta$, where θ substitutes the variables in $\boldsymbol{x}$ with other terms. However, it fails to capture the concept *"anything except"*, which is crucial in expressing the meaning for non-ground negative subgoals. Chan proposed that the computed answers could be in the form of an equality formula under the semantics of *Clark's Equational Theory* (CET) [8, 27], and he called it *constructive negation* answer [4, 5, 37]. If an equality formula $\forall(E)$ is the answer to the query $p(\boldsymbol{t})$, then $\forall(\neg E)$ is the constructive negation for the query $\neg p(\boldsymbol{t})$ [5].

A binding substitution can not be negated, while a constructive negation answer can because it is an equality formula. A constructive negation answer can thus be used to represent the answer for a negative subgoal. Constructive negation answer is thus more expressive than the traditional one in this sense [37].

Throughout this dissertation we use constructive negation answers in the resolutions. For consistency, we use it in both the scenarios where the traditional binding substitution answer is too weak to capture the meaning and where the binding substitution approach is adequate.

1.4.2 Blocking Formulas

Let S be a set of all possible relevant chains whose head is in ρ as follows:

$C1:$ $\quad \rho \leftarrow L_{1,1}, \ldots, L_{1,k_1},$

$\ldots\ldots,$

$Cq:$ $\quad \rho \leftarrow L_{q,1}, \ldots, L_{q,k_q}.$

The *blocking formulas* [50] with respect to the literal ρ of S is defined as a set of all the minimal chains in the following form:

$$\neg\rho \leftarrow \neg L_{1,s_1}, \ldots, \neg L_{q,s_q},$$

where L_{j,s_j} is a literal from the body of jth chain above. Notice that the negated literal could be an equality literal. Each chain in the blocking formulas is said to be *orthogonal* to the set S.

The name *blocking* comes from the fact that ρ could not be derived if $\neg L_{1,s_1} \wedge \cdots \wedge \neg L_{q,s_q}$ has been derived. In case ρ needs to be minimized, not being able to derive it means deriving its negation. This is exactly what each blocking formula intends to do. It is thus easy to see that a nonmonotonic mechanism has already been built into the blocking formulas.

1.4.3 Circumscription-Based Predicate Completion

The *predicate completion* [8] has been used to minimize the extensions of certain predicates, and the basic idea behind the predicate completion is that the only objects that satisfy a predicate are those that must do so.

Given a set S of predicates, we will use $pos(S)$, $neg(S)$ and $any(S)$ to denote the set of all the positive, negative and any literals whose predicates are from S.

Under the semantics of circumscription, each atom p in $pos(P)$ is to be minimized, Z is allowed to vary and Q is fixed. We are first to find the set of all formulas W_i, where W_i is a conjunction of literals in $pos(P) \cup neg(Q)$, such that $CIR(\Pi; P; Z) \models (p \leftarrow W_i)$. Let $\Psi = W_1 \vee \cdots \vee W_n$ for all such W_is. Then the predicate completion of p, based on the circumscriptive formula $CIR(\Pi; P; Z)$, $p \leftrightarrow \Psi$ precisely characterizes the extension of p.

We have found out that such a formula Ψ can be procedurally derived through some classical resolution plus a special selection function which skips the subgoals in $neg(P) \cup any(Q)$. Each W_i in Ψ is a succeeded leaf node of the resolution tree of p. We call Ψ the *procedural definition of p under circumscription*, or the *definition* of p for short. The collection of all such completion formulas (for all p) together with Π constitutes the *procedural definition* for circumscription.

1.5 The Major Contributions of the Dissertation

Now we present what we have accomplished in this dissertation.

1.5.1 CP-resolution

Based on constructive negation and blocking formulas, we introduce *CP-resolution* (Construct P) to search for a sufficient condition under which a negative literal from P can be concluded with respect to a circumscriptive formula.

Given a theory Π and a query $\neg p(\boldsymbol{t})$, CP-resolution can be used to find all formulas Ψ such that $\neg p(\boldsymbol{t}) \leftarrow \neg\Psi$ is valid under the semantics of circumscription.

We have shown that CP-solution is sound with respect to the semantics of circumscription and complete under the the *weakly bounded term size* property, which is weaker than all the other known completion conditions with the existing nonmonotonic reasoning proof procedures.

CP-resolution is developed as a new proof-theoretic framework upon which other proof procedures for nonmonotonic reasoning and logic programming can be developed.

1.5.2 Extracting Answers in Circumscription

CIR-resolution (compute CIRcumscription) is a proof procedure for the semantics of circumscription, based on the CP-resolution. CIR-resolution takes the query in the form of a conjunction of literals, and can be further extended to an arbitrary unquantified formula.

CIR-resolution is basically a classical linear resolution augmented with a CP-resolution nonmonotonic operator. It can extract answers in circumscription. The completion condition for CIR-resolution is weakly bounded term size property, which is inherited from CP-resolution, since the classical resolution is sound and complete. CIR-resolution can return partial answers when the completion condition is not satisfied.

1.5.3 Proof Procedures for Well-founded and Stable Semantics of Logic Programming

With the theoretical foundation laid in Yuan's work [48] where two of the most well known logic programming semantics, *stable* [14] and *well-founded* [12] semantics, are redefined in terms of circumscription of autoepistemic theories, we develop proof procedures for these two semantics on top of the proof-theoretic framework CP-resolution. They are *ST-resolution* (compute STable semantics) and *WF-resolution* (compute Well-Founded semantics).

ST-resolution and WF-resolution can extract answers in logic programs under the stable and well-founded semantics, respectively. Moreover, they inherit all the advantages of CP-resolution, including the weaker completion condition, and the ability to produce partial answers when the completion condition is not met.

1.6 Comparison and Future Work

Let us compare CP-resolution nonmonotonic operator against the negation as failure nonmonotonic operator and its constructive negation enhanced form, as presented in [5]:

1. Negation as failure without constructive negation enhancing, such as the case with SLDNF-resolution [27], has to give up floundered queries, while CP-resolution is designed for negative subgoals with variable.

2. Negation as failure is ordering sensitive and thus may introduce the unnecessary incompleteness due to the improper ordering, while CP-resolution is ordering insensitive.

3. The completion condition for negation as failure is *bounded term size property* [45], while the completion condition for CP-resolution is weakly bounded term size property, and the latter is weaker than the former.

4. CP-resolution uses constructive negation answers, and thus can extract answers from the negative subgoal. In this regard, it is the same as SLD-CNF-resolution [5] and SLSL-resolution [37], but is better than SLDNF-resolution [27], which can only verify the ground negative subgoal.

5. Negation as failure, with or without constructive negation enhancing, is multi-staged, while CP-resolution is one-staged.

6. The answer returned from constructive negation enhanced negation as failure is either nothing or all, because negating a partial answer would lead to unsoundness. On the other hand, CP-resolution produces partial answers, even if the completion condition is not met, because the process of negating is a built-in feature of blocking formulas.

In Section 4.4 we draw a detailed comparison of CIR-resolution against Przymusinski's *query evaluation algorithm* [36]. Although not based on negation as failure, the algorithm basically has all the drawbacks that negation as failure has.

In this dissertation, we pursue a weaker completion condition and the ability to return partial answers when the completion condition is not met, apart from the soundness of the resolutions. We pay very little attention to the efficiency of the algorithm. Therefore, reducing search space of CP-resolution to allow computer implementation is an important future research direction.

Extending the usage of CP-resolution by developing more nonmonotonic proof procedures based on it is a good research topic to work on. One can investigate the feasibility of developing a new proof procedure with our proof-theoretic framework for another important logic programming semantics, *preferential semantics* [9], which is also redefined in terms of circumscription of autoepistemic theories, but is in higher order circumscription [48].

1.7 Organization of the Document

This thesis is divided into six chapters. The preliminaries are presented in Chapter 2. This includes the syntax and semantics of first-order logic, circumscription, Skolem expansion, Clark's equational theories, and constructive negation. The chapter is concluded with some assumptions.

In Chapter 3, we first introduce CP-resolution as a new proof-theoretic framework for nonmonotonic reasoning and then analyze properties of CP-resolution.

Chapter 4 presents CIR-resolution which is based on CP-resolution and can be used to extract answers in circumscription. It is sound with respect to the circumscription semantics and complete under the weakly bounded term size property. A detailed comparison of CIR-resolution with Przymusinski's query evaluation algorithm is also presented.

In Chapter 5, ST-resolution and WF-resolution are developed to extract answers from logic programs with respect to the stable and well-founded semantics, respectively. They are complete under the weakly bounded term size property, and can produce partial answers when the completion conditions are not met.

Chapter 6 gives the conclusions.

The appendices contain some detailed information regarding the motivation and insight of the definition of the resolutions, and the possible simplification. They also contain some complicated proofs.

Chapter 2

Preliminary

Most of the concepts covered in this chapter can be found in books [32, 6, 17, 27, 2].

2.1 Syntax of the Logic Theories

A *first order theory* consists of an *alphabet*, a *first order language* (or a *language* for short), a set of *axioms* and a set of *inference rules*. The language consists of the *well-formed formula (wff)* of the theory. The axioms are a designated subset of *wff*. The axioms and *rules of inference* are used to derive the *theorems* of the theory.

Definition 2.1 An alphabet consists of seven classes of symbols:

(a) variable;

(b) constant;

(c) functions symbols;

(d) predicate symbols;

(e) logic connectives;

(f) quantifiers;

(g) punctuation symbols.

□

Variables are denoted as x, y and z (possibly subscripted). We use $\boldsymbol{x}$ (resp. $\boldsymbol{y}$ and $\boldsymbol{z}$) to denote an n-tuple of variables $x_1, \ldots, x_n$ (resp. $y_1, \ldots, y_n$ and $z_1, \ldots, z_n$). Constants are denoted as a, b and c (possibly subscripted). We use $\boldsymbol{a}$ (resp. $\boldsymbol{b}$ and $\boldsymbol{c}$) to denote an n-tuple of constants $a_1, \ldots, a_n$ (resp. $b_1, \ldots, b_n$ and $c_1, \ldots, c_n$).

Function symbols of various arities > 0 will be denoted as f and g (possibly subscripted). We use $\boldsymbol{f}$ (resp. $\boldsymbol{g}$) to denote a tuple of function symbol $f_1, \ldots, f_n$ (resp. $g_1, \ldots, g_n$). Constants are by no means an indispensable member of an alphabet, since a constant can be expressed as an 0-place function symbol. Predicate symbols are denoted as p, q and z (possibly subscripted). The 0-place predicate symbol is a *proposition*. A special predicate symbol *equality* is denoted as e. We use P, Q and Z (possibly subscripted) to denote collections of predicate symbols with same properties. Let S be a set of predicates. The $pos(S)$, $neg(S)$ and $any(S)$ are used to denote the set of all positive, negative and any literal whose predicates are in S, respectively. The logic connectives are $\neg$, $\wedge$, $\vee$, $\leftarrow$, $\rightarrow$ and $\leftrightarrow$, while the quantifies are $\exists$ and $\forall$. Finally the punctuation symbols are "(", ")" and ",". A *propositional logic* is a reduction of a first order logic whose language contains only propositional symbols, logic connectives and punctuation symbols.

We mainly consider three categories of predicates, which are P, Z and Q. E is used to denote the collection of equality symbols, and it is part of Q.

Definition 2.2 A *term* is defined inductively as follows:

(a) A variable is a term;

(b) A constant is a term;

(c) If f is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

□

We use $\boldsymbol{t}$ (resp. $\boldsymbol{s}$) to denote an n-tuple term $t_1, \ldots, t_n$ (resp. $s_1, \ldots, s_n$), and call it an n-tuple term.

Definition 2.3 A *literal* is an atom or its negation. □

Definition 2.4 Let $\boldsymbol{s} = s_1, \ldots, s_n$ and $\boldsymbol{t} = t_1, \ldots, t_n$ be n-tuple terms. An *n-fold equality* $e(\boldsymbol{s}, \boldsymbol{t})$, or equivalently $e(\langle s_1, \ldots, s_n \rangle, \langle t_1, \ldots, t_n \rangle)$, is defined as $e(s_1, t_1) \wedge \cdots \wedge e(s_n, t_n)$, and its negation $\neg e(\boldsymbol{s}, \boldsymbol{t})$, or equivalently $\neg e(\langle s_1, \ldots, s_n \rangle, \langle t_1, \ldots, t_n \rangle)$, is called an *$n$-fold inequality*, is defined a $\neg e(s_1, t_1) \vee \cdots \vee \neg e(s_n, t_n)$. □

We do not distinguish an n-fold equality from an 1-fold equality, and use the same predicate symbol for whatever n, $n \geq 1$. Both equality and inequality are deemed as literals. A literal is called an *equality literal* if it is an equality or an inequality,

otherwise it is a *nonequality literal.* An *equality formula* is defined as a formula containing equality literals only.

A *sequence of literals*, $L_1, \ldots, L_n$, ordered from left to right, together with an equality formula E, represents

$$E \wedge L_1 \wedge L_2 \wedge \cdots \wedge L_n.$$

Let C be a sequence of literals, then a *goal* is a formula of the form $\leftarrow C$ and a *chain*, which is also called a *rule* in logic programming if the head is positive, is a formula of the form $L_0 \leftarrow C$, where $L_0 \notin E$. For simplicity, we may also denote a goal and a chain using

$$\leftarrow L_1, L_2, \ldots, L_n,$$

and

$$L_0 \leftarrow L_1, L_2, \ldots, L_n,$$

respectively.

A chain $L_i \leftarrow L_1, L_2, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n, \neg L_0$ is a *variant* of the chain $L_0 \leftarrow L_1, L_2, \ldots, L_n$, for $1 \leq i \leq n$ such that $L_i \notin E$.

Note the head of a chain can be either positive or negative. There are two types of literals in a goal, namely, *framed* or otherwise. A framed literal is a literal L with a special notation, $[L]$. It does not play any logical role, but is used for recording the literal which is resolved. A *cell* is a collection all of unframed literals which are not separated with framed literals.

A goal is *pseudo-empty* if it contains no nonequality literals, and its equality formula is satisfiable. An empty goal is a trivial pseudo-empty goal.

2.2 Interpretations and Models

Without semantics, a formula is a grammatically correct sequence of symbols. Semantics represents the meaning the sequence conveys.

Definition 2.5 An *interpretation* I of a language L consists of the following:

(a) A non-empty set $|I|$, called the *universe* of the interpretation;

(b) For each n-place function symbol f in L, the assignment of a mapping f^I from $|I|^n$ to $|I|$;

(c) For each n-place predicate symbol p in L, the assignment of a mapping p^I from $|I|^n$ to $\{TRUE, FALSE\}$, (or, equivalently, a relation on $|I|^n$).

□

We use α or β (possibly subscripted) to denote an object in an universe, $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$ (possibly subscripted) to denote an n-tuple of objects in a universe.

Definition 2.6 Let I be an interpretation of a language L. Let ξ be a function, called *variable assignment*, with a domain that includes all the variables in L and with a range that is a subset of $|I|$. For each term t or formula Φ that can be constructed within L, we define a value $I(t)_\xi$ in $|I|$, called the *value* of t, or $I(\Phi)_\xi$ in $\{TRUE, FALSE\}$, called the *truth value* of Φ, as follows:

1. (a) If $t_1, \ldots, t_n$ are terms and f is an n-place function symbol in L, then $I(f(t_1, \ldots, t_n))_\xi = f^I(I(t_1)_\xi, \ldots, I(t_n)_\xi)$;
   (b) If $t_1, \ldots, t_n$ are terms and p is an n-place predicate symbol in L, then the truth value $I(p(t_1, \ldots, t_n))_\xi$ is obtained by calculating the value of $p^I(I(t_1)_\xi, \ldots, I(t_n)_\xi)$;
   (c) For a variable x in L, $I(x)_\xi$ is $\xi(x)$;
2. (a) $I(\Phi \vee \Psi)_\xi$ is true if either $I(\Phi)_\xi$ or $I(\Psi)_\xi$ is true, otherwise it is false;
   (b) $I(\Phi \wedge \Psi)_\xi$ is true if both $I(\Phi)_\xi$ and $I(\Psi)_\xi$ are true, otherwise it is false;
   (c) $I(\neg\Phi)_\xi$ is true if $I(\Phi)_\xi$ is false;
3. (a) $I(\forall x\Phi)_\xi$ is true if $I(\Phi)_{\xi<x/\alpha>}$ is true for each object $\alpha \in |I|$, otherwise it is false, where $\xi < x/\alpha >$ is a variable assignment ξ' that is identical to ξ except that $\xi'(x) = \alpha$;
   (b) $I(\exists x\Phi)_\xi$ is true if $I(\Phi)_{\xi<x/\alpha>}$ is true for some object $\alpha \in |I|$, otherwise it is false, where $\xi < x/\alpha >$ is the same as in 3. (a).

□

We write $I \models \Phi$ if $I(\Phi)$ is true, and $I \not\models \Phi$ otherwise, where Φ is a sentence.

Definition 2.7 Let I be an interpretation for a language L, and Φ be a formula in L. Φ is *satisfiable* in I if $I \models \exists(\Phi)$; Φ is *valid* in I if $I \models \forall(\Phi)$; Φ is *unsatisfiable* in I if $I \not\models \exists(\Phi)$. □

Definition 2.8 Let Π be a sentence in a language L. Then an interpretation I of L is a *model* for Π if $I \models \Pi$. Π is *satisfiable* if it has at least one model, otherwise it is *unsatisfiable*; it is *valid* if every interpretation I of L is a model of Π. □

Definition 2.9 Let Φ be a set of sentences and Ψ be a sentence of a language L. We say Ψ is a *logical consequence* of Φ, or Ψ is *logically implied* by Φ, or Ψ is *entailed* by Φ, denoted as $\Phi \models \Psi$, if for every interpretation I of L, I is a model for Φ implies that I is a model for Ψ. □

We also say Φ is *stronger than* Ψ if $\Phi \models \Psi$, and Ψ is *weaker* than Φ otherwise.

Proposition 2.1 *Let Φ and Ψ be sentences. $\Phi \models \Psi$ if and only if $\Phi \cup \neg\Psi$ is unsatisfiable [2, 32, 27, 6].* □

Definition 2.10 A *theory* is a set of sentences closed under logic consequences, a member of the theory is called a *theorem*. A theory T is *complete* if for every sentence Ψ, either Ψ or $\neg\Psi$ is a theorem of T. □

Definition 2.11 Let Φ and Ψ be formulas of a language L. We say Φ and Ψ are *logically equivalent*, denoted as $\Phi \equiv \Psi$, if $\forall(\Phi \leftrightarrow \Psi)$ is valid. □

Compactness theorem states that a set of sentences Π is unsatisfiable only if some finite subset of Π is unsatisfiable [32, 2, 6].

Let T be a theory. A *proof* of a sentence Φ from T is a classification of Φ as a theorem of T. A *theorem prover* or *proof procedure* is a mechanism to carry on such proofs. The theorem prover is *sound* if the proof of a sentence Φ from T implies Φ is a theorem of T. A theorem prover is complete if for every theorem Φ of T there is a proof of Φ from T. We say T is *decidable* if there exists a mathematical method for determination whether there is a proof of Φ from T for any given sentence Φ. Otherwise, it is undecidable. There exists a sound and complete theorem prover for any first order theory. But in general, first order logic is undecidable. Due to the fact that every complete theory is decidable, first order logic is sometimes said to be *semidecidable* [32, 2].

Definition 2.12 Let L and L' be languages such that L' is an expansion of L. Let I_1 and I_2 be interpretations of L and L', respectively. Then I_2 is an *expansion* of I_1 to L', while I_1 is a *reduction* of I_2 to L, if (a) $|I_1| = |I_2|$; (b) I_1 and I_2 assign exactly the same mapping for each function and predicate symbol in L. □

2.2.1 Free Universes and Free Interpretations

Definition 2.13 A *ground term* is a term not containing variables. Similarly, a *ground atom* is an atom not containing variables. □

Definition 2.14 Let L be a language. The *free universe* U_L for L is the set of all the ground terms, which can be formed out of the constants and function symbols appearing in L. (In case that L has no constants, we add a constant to L so that to form ground terms.) The *free base* B_L for L is the set of all ground atoms which can be formed by using predicate symbols from L with ground terms in U_L. □

In literature, a free universe is sometimes called a *Herbrand universe*, and a free base a *Herbrand base*.

Definition 2.15 Let L be a language. A *free interpretation* I for L is an interpretation which satisfies the following conditions:

1. $|I|$ is the free universe U_L;
2. Constants in L are assigned to themselves in U_L;
3. If f is an n-place function symbol in L, then the mapping from $(\mathsf{U}_\mathsf{L})^n$ to U_L defined by $(t_1, \ldots, t_n) \rightarrow f(t_1, \ldots, t_n)$ is assigned to f;
4. Each ground atom is assigned either *TRUE* or *FASLSE*.

□

Another informal definition of a free interpretation is that it is a subset of the free base.

Free interpretation uses syntactic way to handle a formally semantical issue. A free interpretation I *syntactically* interprets all the function symbols in L. Because of this, we do not have to distinguish p^I from p for each predicate symbol p, and f^I from f for each function symbol f in L.

Definition 2.16 Let Ψ be a sentence in a language L. A free interpretation I of L is a *free model* of Ψ if it is a model of Ψ. □

The assumption of the underlying language, with respect to a given sentence Π, is an important issue, because an underlying language uniquely defines the underlying free universe. Let Π be a sentence. The language L_Π is a language defined by the

constants, function symbols and predicate symbols appearing in Π. Understandably, a legitimate underlying language L must be an expansion of the language L_Π.

Some researchers assume that L_Π is exactly the underlying language [44, 27]. Another extreme is found in [23] where Kunen assumes that the underlying language L is an expansion of L_Π such that for each $n \geq 0$ there are infinitely many n-place function symbols in L. While in [12], van Gelder *et al.* suggests that the underlying language L is L_Π plus one more symbol which serves as a constant, function and predicate symbol simultaneously.

2.3 Deletion Rules

By using the following deletion rules we can delete some irrelevant clauses from the given ground set Π of clauses, and the remaining simplified sentence is unsatisfiable if and only if Π is unsatisfiable [7, 17].

Tautology Rule Delete all the ground clauses from Π which are tautologies. The remaining set Π' is unsatisfiable if and only if Π is unsatisfiable.

One-Literal Rule For each unit clause L in Π, delete L from Π, as well as the literal $\neg L$ from each clause in Π. The remaining set Π' is unsatisfiable if and only if Π is unsatisfiable.

Pure-Literal Rule If the $\neg L$ is not contained in any clause in Π, delete each clause which contains the literal L. The remaining set Π' is unsatisfiable if and only if Π is unsatisfiable.

Splitting-Rule Rewrite Π in the form: $\Pi = (\Pi_1 \vee L) \wedge (\Pi_2 \vee \neg L) \wedge \Pi_3$, where Π_1, Π_2 and Π_3 are sets of clauses containing neither L nor $\neg L$. Then both $\Pi_1 \wedge \Pi_3$ and $\Pi_2 \wedge \Pi_3$ are unsatisfiable if and only if Π is unsatisfiable.

2.4 Skolem Expansions

Definition 2.17 A quantifier free formula is said in *conjunctive normal form* if it is a conjunction of disjunction of literals. It is in *disjunctive normal form* if it is a disjunction of conjunction of literals. □

Every quantifier free formula has at least one conjunctive normal form and one disjunctive normal form [25].

Definition 2.18 A formula Φ is in *prenex conjunctive normal form* if it is in the form $(Q_1x_1)\ldots(Q_nx_n)\Psi$, where $(Q_1x_1)\ldots(Q_nx_n)$ is called *prefix* of Φ, each Q_i is either an universal or existential quantifier, while Ψ, called the *matrix* of Φ, is a quantifier free formula in conjunctive normal form. □

Every formula is logically equivalent to a formula in prenex conjunctive normal form [27]. We can reasonably assume that a given sentence is in prenex normal form.

Definition 2.19 A sentence in prenex normal form is *universal* if all of its quantifiers are universal; it is *existential* if all of its quantifiers are existential. □

Proposition 2.2 *A universal sentence* Π *in* L *has a model if and only if it has a free model.* □

A universal sentence is equivalent to a set of clauses. A universal sentence which is in the form of a set of clauses is said to be in *clausal form.* The *ground instantiation* of a set of clauses is the collection of the ground instances of each clause in the set.

Let Φ be a sentence in a language L such that $\Phi = (Q_1x_1)\ldots(Q_nx_n)\Psi$ is in prenex normal form. Suppose (Q_ix_i) is an existential quantifier in the prefix, $1 \leq i \leq n$, and $Q_{s1},\ldots,Q_{sm}$ are all the universal quantifiers appearing before Q_i (probably none), $1 \leq s1 < s2 < \cdots < sm < i$, we then use a new m-place function symbol f distinct from all other function symbols, called a *Skolem function*, to replace all x_i in Ψ by $f(x_{s1},\ldots,x_{sm})$, and delete Q_i from the prefix. The resulted sentence without containing existential quantifier is called a *Skolem normal form* of Φ. The process to convert Φ into its Skolem normal form is called *Skolemization* of Φ. The expansion of language L by Skolem function symbols, called the *Skolem expansion* of L, is defined as:

$$L^s = L \cup \{f(x_{s1},\ldots,x_{sm}) \mid \ f \text{ is a Skolem function.} \ \}$$

Proposition 2.3 *A sentence is satisfiable if and only if its Skolem normal form is satisfiable [25, 7, 17, 27, 29].* □

Note, in general, a sentence does not logically equivalent to its Skolem normal form.

2.5 Unification

Definition 2.20 A *substitution* θ is a finite set of the form $\{x_1/t_1, \ldots, x_n/t_n\}$, where each x_i is a variable, each t_i is a term distinct from x_i, and the variables $x_1, \ldots, x_n$ are distinct. Each element x_i/t_i is called a *binding* for x_i. θ is called a *ground substitution* if each t_i is a ground term. It is called a *variable pure substitution* if each t_i is a variable. The substitution which is an empty set is called the *identity substitution.* □

Let $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ be a substitution. We call the set of variables $\{x_1, \ldots, x_n\}$ the *domain* of θ. We call the set of variables appearing in $\{t_1, \ldots, t_n\}$ the *range* of θ.

Definition 2.21 An *expression* is either a term, a literal or a conjunction or disjunction of literals. A *simple expression* is either a term or an atom. □

Let $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ be a substitution and κ be an expression. Then $\kappa\theta$, the instance of κ by θ, is the expression obtained from κ by simultaneously replacing each occurrence of the variable x_i in κ by the term t_i $(i = 1, \ldots, n)$.

Definition 2.22 Let $\theta = \{x_1/s_1, \ldots, x_m/s_m\}$ and $\sigma = \{y_1/t_1, \ldots, y_n/t_n\}$ be substitutions. Then the *composition* $\theta\sigma$ of θ and σ is the substitution obtained from the set $\{x_1/s_1\sigma, \ldots, x_m/s_m\sigma, y_1/t_1, \ldots, y_n/t_n\}$ by deleting any binding $x_i/s_i\sigma$ for which $s_i = s_i\sigma$ and deleting any binding y_j/t_j for which $y_j \in \{x_1, \ldots, x_m\}$. □

Substitutions have properties indicated by the following proposition [27, 7, 29].

Proposition 2.4 *Let θ, σ and γ be substitutions and ε be the identity substitution. Then*

1. $\theta\varepsilon = \varepsilon\theta = \theta$;
2. $(\kappa\theta)\sigma = \kappa(\theta\sigma)$, *for all expressions* κ;
3. $(\theta\sigma)\gamma = \theta(\sigma\gamma)$.

□

Definition 2.23 Let θ and σ be substitutions. θ is said *more general* than σ if there exists another substitution γ such that $\theta\gamma = \sigma$. □

Definition 2.24 Let κ be an expression and V be the set of variables occurring in κ. A *renaming substitution* for κ is a variable pure substitution $\{x_1/y_1, \ldots, x_n/y_n\}$ such that $\{x_1, \ldots, x_n\} \subseteq V$, the y_i are distinct and $(V - \{x_1, \ldots, x_n\}) \cap \{y_1, \ldots, y_n\} = \emptyset$. □

Definition 2.25 Two expressions κ_1 and κ_2 are *variants* if there exist substitutions θ_1 and θ_2 such that $\kappa_1\theta_1 = \kappa_2$ and $\kappa_1 = \kappa_2\theta_2$. □

Definition 2.26 Let S be a finite set of simple expressions. A substitution θ is a *unifier* for S if $S\theta$ is a singleton. A unifier θ is called a *most general unifier* (*mgu*) of S if for each unifier σ of S, θ is more general than σ. □

Definition 2.27 A substitution θ is *idempotent* if $dom(\theta) \cap ran(\theta) = \emptyset$. □

A substitution θ is idempotent if and only if $\theta\theta = \theta$. θ is a *mgu* of S and θ is idempotent, if and only if for every unifier σ of S, we have $\sigma = \theta\sigma$ [27, 24].

2.6 Clark's Equational Theory (CET)

Intuitively, the equality e is understood as *identity*. However, a first-order interpreter would interpret it as an arbitrary 2-place relation if there is no extra axiom to enforce its interpreting into an identity relation.

Reiter [40] introduced the term *unique name assumption* (*UNA*) which states that distinct names denote distinct objects in the domain of discourse. In terms of equality, $e(a, b)$ is false if a and b are distinct names. Extra axioms are needed to enforce *UNA*.

Definition 2.28 The following eight axioms are called *Clark's equational theory* (CET) [8, 27, 23, 35]:

CET1. $e(x, x)$;

CET2. $e(x, y) \rightarrow e(y, x)$;

CET3. $e(x_1, x_2) \wedge e(x_2, x_3) \rightarrow e(x_1, x_3)$;

CET4. $e(x_1, y_1) \wedge \cdots \wedge e(x_m, y_m) \rightarrow e(f(x_1, \ldots, x_m), f(y_1, \ldots, y_m))$, *for any function* f;

CET5. $e(x_1, y_1) \wedge \cdots \wedge e(x_m, y_m) \rightarrow (p(x_1, \ldots, x_m) \rightarrow p(y_1, \ldots, y_m))$, *for any predicate* p;

CET6. $\neg e(f(x_1,\ldots,x_m), g(y_1,\ldots,y_n))$, *for any two different function symbols* f and g;

CET7. $e(f(x_1,\ldots,x_m), f(y_1,\ldots,y_m)) \rightarrow e(x_1,y_1) \wedge \cdots \wedge e(x_m,y_m)$ *for any function symbols* f;

CET8. $\neg e(\tau(x), x)$ *for any term* $\tau(x)$ *different from* x, *but containing* x.

All the axioms in CET are universally quantified. □

CET1 through CET3 indicate that CET is an equivalence relation. CET4 and CET5 are *substitutivity* of CET. CET1 through CET5 overall are *equality axioms* which enforce the equality to be interpreted as identity, while the remaining three are called *freeness axioms* or *unique name axioms.*

The freeness axioms are language dependent [44]. We will use CET_{L} to denote the specific CET determined by the language L.

Let L be a language, U_{L} and CET_{L} be the free universe and CET determined by L, respectively. We have the following lemma,

Lemma 2.5 *An interpretation* I *is a model of* CET_{L} *if and only if there exists a free model* I' *of* CET_{L} *over* U_{L} *which is isomorphically embedded in* I *[23].* □

Lemma 2.6 ([8, 27]) *Let* $p(t_1,\ldots,t_n)$ *and* $p(s_1,\ldots,s_n)$ *be atoms.*

1. *If* $p(t_1,\ldots,t_n)$ *and* $p(s_1,\ldots,s_n)$ *are not unifiable, then* $\neg\exists(e(t_1,s_1) \wedge \cdots \wedge e(t_n,s_m))$ *is a logical consequence of* CET;
2. *If* $p(t_1,\ldots,t_n)$ *and* $p(s_1,\ldots,s_n)$ *are unifiable with mgu* $\theta = \{x_1/a_1,\ldots,x_k/a_k\}$, *then* $\forall(e(t_1,s_1) \wedge \cdots \wedge e(t_n,s_m) \leftrightarrow e(x_1,a_1) \wedge \cdots \wedge e(x_k,a_k))$ *is a logical consequence of* CET.

□

An inequality $\neg e(t_1,t_2)$ is *valid* if t_1 and t_2 cannot be unified. It is *satisfiable* if there exists a substitution θ such that $\neg e(t_1,t_2)\theta$ is valid. Similarly, an equality $e(t_1,t_2)$ is *valid* if t_1 and t_2 are syntactically identical. It is *satisfiable* if t_1 and t_2 are unifiable [4, 5, 37, 24].

Let $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ be ground n-tuple of terms in L, then either $\text{CET} \models e(\boldsymbol{t}_1,\boldsymbol{t}_2)$ if $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are syntactically identical, or $\text{CET} \models \neg e(\boldsymbol{t}_1,\boldsymbol{t}_2)$ otherwise [27]. For a free interpretation I over the free universe U_{L} to be a model of CET_{L}, it must contain $e(\boldsymbol{t},\boldsymbol{t})$ for every n-tuple of terms of L, as well as $\neg e(\boldsymbol{t}_1,\boldsymbol{t}_2)$ for every syntactically distinct n-tuples of terms $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ of L.

CET is language sensitive. It has no effect on Skolem terms, which are in the Skolem expansion L^s of the underlying language L.

2.7 Clark's Predicate Completion

Clark's predicate completion [8] is a commonly used declarative semantics for logic programs and deductive databases [27, 8]. The intended meaning behind it is that the sufficient condition for a certain property is also its necessary condition. The set of clauses provides the "*if*" half of the *definition* for each predicate, which should be completed with its "*only if*" half. We assume the given sentence is universal or a set of clauses.

Let Π be a set of clauses in L, which is also called a *database*, and P be a set of predicate symbols in L. Let a clause in Π be in the following form:

$$p(\boldsymbol{t}) \leftarrow L_1, \ldots, L_m$$

where p is a predicate symbol from P and $\boldsymbol{t}$ is an n-tuple term. Then the clause can be rewritten into its *Clark's general form*:

$$\forall \boldsymbol{x}(p(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}(e(\boldsymbol{x}, \boldsymbol{t}) \wedge L_1 \wedge \cdots \wedge L_m)),$$

where $\boldsymbol{x}$ is an n-tuple of variables not occurring in the original clause, $e(\boldsymbol{x}, \boldsymbol{t})$ is n-fold equality, and $\boldsymbol{y}$ is a tuple of variables in the original clause. In case t_i is a variable term, delete $e(x_i, t_i)$ and use x_i to replace t_i in the Clark's general form clause.

Suppose there are exactly k clauses in Π whose head is an atom from predicate p, and the general form for these k clauses are:

$$p(\boldsymbol{x}) \leftarrow W_1,$$
$$\cdots\cdots$$
$$p(\boldsymbol{x}) \leftarrow W_k,$$

where each W_i is a conjunction of literals. Then the "only if" half of the *definition* of p in Π is:

$$\forall \boldsymbol{x}(p(\boldsymbol{x}) \rightarrow W_1 \vee \cdots \vee W_k),$$

and the completed definition of p in Π is:

$$\forall \boldsymbol{x}(p(\boldsymbol{x}) \leftrightarrow W_1 \vee \cdots \vee W_k).$$

The empty disjunction identifies *FALSE*, so when there is no clause in Π whose head is a atom from p, the completed definition of p in Π is simply $\forall \boldsymbol{x}(\neg p(\boldsymbol{x}))$. Observe, each variable in $W_1 \vee \cdots \vee W_k$ but not in $\boldsymbol{x}$ becomes existentially quantified in the "*only if*" half of the definition through predicate completion.

The completion of P in Π is defined as:

$$\mathrm{COMP}(\Pi; P) = \Pi \cup \{\forall \boldsymbol{x}\,(p(\boldsymbol{x}) \rightarrow W_1 \vee \cdots \vee W_k) \mid p \in P\} \cup \mathrm{CET}.$$

The CET involved in COMP(Π; P) is determined by the underlying language of Π.

2.8 Constructive Negation Answers

We use SEM(Π) to denote the declarative semantics of a given sentence Π with respect to a certain formalism. Model theoretical approach to the declarative semantics usually uses one canonical model, or a set of preferred models of Π to describe the meaning of SEM(Π). By SEM(Π) $\models \Psi$, where Ψ is a sentence, we mean an interpretation I is a model of SEM(Π) (a preferred model of Π) implies that it is a model of Ψ.

The role of an intelligence machine is to simulate a reasoning agent, the database Π represents the agent's knowledge regarding the real world. When given a query Φ, the intelligent machine is supposed to give a YES (resp. NO) answer if SEM(Π) $\models \exists\Phi$ is true (resp. false). In this approach, the query is assumed to be existentially quantified.

A more sophisticated approach is that the intelligent machine is capable of giving an answer ANS of some kind such that SEM(Π) $\models \forall(\mathrm{ANS} \rightarrow \Phi)$ is true. We say the latter approach *extracts answers* in the semantics SEM. The answer ANS may give the details of 'What', 'Which', 'When', 'How', etc. A *query* Ψ in this approach is a quantifier free first order formula. A *goal* is a formula in the oriented form $\leftarrow \Psi$, where Ψ is a query.

Traditionally, an answer is in the form of a binding substitution θ as in PROLOG logic programming [27]. Given a program Π and a query Ψ, a resolution turns out a composition of *mgu's* θ, implying that SEM(Π) $\models \forall \boldsymbol{x}(\Psi\theta)$, or equivalently SEM($\Pi$) $\cup$ CET $\models \forall \boldsymbol{x}(e(\boldsymbol{x}, \boldsymbol{x}\theta) \rightarrow \Psi)$, where $\boldsymbol{x}$ is a tuple of free variables in Ψ. This kind of answer can express the concept "*all*", which is typically an answer to a positive query, but fails to capture the concept "*anything except*", typically conceived in a negative query, and therefore has limited expressibility.

Let Qr be a query with a tuple of free variables $\boldsymbol{x}$. A *constructive negation* answer, proposed by Chan [4, 5, 37], is a universal sentence $\forall(E \rightarrow Qr)$, where E is an equality formula. We would prefer that the equality formulas contains no Skolem term due to its undecidability with $\mathrm{CET}_{\mathrm{L}}$. The constructive negation answer is much more expressive than the traditional one [37].

According to Chan [5], the answer to the query $\neg p(\boldsymbol{x})$ should be $\forall \boldsymbol{x}\,(\neg E)$, if the answer to the query $p(\boldsymbol{x})$ is $\forall \boldsymbol{x}\,(E)$. To achieve this, E has to be in a form which

can be negated. A binding substitution can not be negated, while a constructive negation answer can, because it is an equality formula. It thus makes it possible to extract answers from negative subgoals.

With free approach, where we are concerned with only free models, constructive negation answer is language sensitive, since the universe is determined by the underlying language, assuming CET. For instance, $\forall x \neg e(x, a)$ is analogous to *FALSE* with respect to a language in which the only function symbol is a, but has numerous models in any language containing function symbols other than a. No matter what the underlying language is, the implication CET $\models \forall(\text{ANS} \rightarrow Qr)$ is always true and language independent. In other words, the user has the privilege to interpret the answer on the basis of his/her knowledge regarding the universe. For the above example, if the user believes there is nothing other than a, then $\forall \neg e(x, a)$ is equivalent to "*nothing*", otherwise it could mean anything except a. The constructive negation answer is thus *user-oriented.*

2.9 Circumscription

The definition of circumscription can be given in both syntactic and model theoretic ways, which are proved to be coincident with each other [26, 10].

Let $\Pi(Q, P, Z)$ be a first-order sentence, where Q, P, Z are partitions of all predicates in Π. Then $CIR(\Pi; P; Z)$ denotes the circumscription of Π on P with Z varying. More formally, the syntactical definition of $CIR(\Pi; P; Z)$ is,

$$CIR(\Pi; P; Z) = \Pi(Q, P, Z) \wedge \neg \exists P', Z' (\Pi(Q, P', Z') \wedge P' \prec P),$$

where $P' \prec P$ stands for that the interpretation of P' is a proper subset of that of P.

The second conjunct

$$\neg \exists P', Z'(\Pi(Q, P', Z') \wedge P' \prec P)$$

is called *circumscription axiom* of the theory $\Pi(Q, P, Z)$. Circumscription axiom plays the crucial role of nonmonotonicity. The circumscription axiom is a second order formula because of the predicate variables it contains.

Let M_1 and M_2 be two models of Π. We say $M_1 \leq M_2$ mod (P, Z) if M_1 and M_2 differ only in how they interpret predicates from P and Z, and for every predicate from P, the relation assigned by M_1 is a subset of that assigned by M_2. A model M_1 of Π is (P, Z)-*minimal* if $M_2 \leq M_1$ mod (P, Z) implies $M_1 \leq M_2$ mod (P, Z) for every model M_2 of Π [26, 15]. This is the model theoretical definition of circumscription.

An interpretation M is a model of $CIR(\Pi; P; Z)$ if and only if it is a (P, Z)-minimal model of $\Pi(Q, P, Z)$ [31, 30, 26].

Let a *P-completing* formula with respect to Π denote a formula of the form

$$p(\boldsymbol{a}) \rightarrow \bigvee_i D_i,$$

where $\Pi \models p(\boldsymbol{a}) \leftarrow D_i$, $p(\boldsymbol{a}) \in pos(P)$, D_i is a finite conjunction of literals $L_j \in (neg(P) \cup any(Q))$, and $\bigvee_i D_i$ may represent an infinite disjunction.

Theorem 2.7 *Let $\Pi(Q, P, Z)$ be a first-order sentence. Then*

$$CIR(\Pi; P; Z) \equiv \Pi \wedge D_P,$$

where D_P is the conjunction of the P-completing formulas for all $p(\boldsymbol{a})$, $p(\boldsymbol{a}) \in pos(P)$. □

Proof: $\Rightarrow$ Assume $CIR(\Pi; P; Z) \not\models p(\boldsymbol{a}) \rightarrow \bigvee_i D_i$. There must exist a model M of $CIR(\Pi; P; Z)$ such that $M \models p(\boldsymbol{a})$, but $M \not\models \bigvee_i D_i$. Let Q_M denote the conjunction of all the ground literals L_k such that $L_k \in any(Q)$ and $M \models L_k$, $neg(P)_M$ denote the conjunction of all the ground literals L_k such that $L_k \in neg(P)$ and $M \models L_k$.

$$\Pi \models p(\boldsymbol{a}) \leftarrow neg(P)_M \wedge Q_M \tag{2.1}$$

must be true. Otherwise, there must exist a model M' of Π such that $M' \models neg(P)_M \wedge Q_M$, but $M' \not\models p(\boldsymbol{a})$. This implies that $M' \leq M$ mod (P, Z) but not
$M \leq M'$ mod (P, Z). M is therefore not a model of $CIR(\Pi; P; Z)$. By Formula (2.1),

$$\Pi \cup neg(P)_M \cup Q_M \cup \{\neg p(\boldsymbol{a})\}$$

is unsatisfiable. By *compactness theorem* [32, 2, 6], there must exist a finite set D' of literals such that

$$D' \subseteq neg(P)_M \cup Q_M$$

and

$$\Pi \cup D' \cup \{\neg p(\boldsymbol{a})\}$$

is unsatisfiable. $M \models D'$ and $\Pi \models p(\boldsymbol{a}) \leftarrow D'$ follow immediately. Since $D' \in neg(P) \cup Q$, it must be some D_j in $\bigvee_i D_i$. Therefore $M \models \bigvee_i D_i$, which in turn contradicts the assumption.
$\Leftarrow$ Assume there is a model M of Π such that $M \models \Pi \wedge D_P$, but $M \not\models CIR(\Pi; P; Z)$. There must exist a model M' of Π such that $M' < M$. Let $p(\boldsymbol{a})$ be the ground atom,

$p(\boldsymbol{a}) \in pos(P)$, such that $M \models p(\boldsymbol{a})$, but $M' \models \neg p(\boldsymbol{a})$. Since $M \models \Pi \wedge D_P$, $M \models p(\boldsymbol{a}) \rightarrow \bigvee_i D_i$ holds, where $p(\boldsymbol{a}) \rightarrow \bigvee_i D_i$ is the P-completing formula. $M \models \bigvee_i D_i$ thus holds. There must exist an i such that $M \models D_i$. By definition $D_i \in neg(P) \cup any(Q)$, therefore $M' \models D_i$. Otherwise $M' < M$ doesn't hold. By definition $\Pi \models p(\boldsymbol{a}) \leftarrow D_i$, $M' \models p(\boldsymbol{a})$ therefore must hold. This contradict the assumption that $M' \models \neg p(\boldsymbol{a})$. □

Lots of efforts and studies have been given to the issue of the collapsibility of circumscription. However in general, circumscription does not collapse to a first-order sentence [26, 17, 39, 21].

2.10 Assumptions

In this dissertation, we assume the following.

ASSUME CLARK'S EQUATIONAL THEORY (CET). The users are only interested in the objects which can be identified by their names, and tend to acknowledge that different names represent different objects in the world. It is thus quite natural for us to assume unique name assumption. The special predicate equality is thus introduced, and the equality axioms are introduced to enforce the equality to be interpreted as *identity*. In the sequel, by a theory Π, we actually mean $\Pi \cup$ CET. By an equality formula being satisfiable, we mean it is satisfiable with CET.

ASSUME THAT THE GIVEN THEORY IS A CONSISTENT UNIVERSAL SENTENCE WITHOUT EQUALITY LITERALS. The universal sentences, or a set of clauses, are of great interest to the researchers in AI areas. Usually the databases are described with sets of clauses. Moreover, as Etherington *et al.* pointed out that there exists some consistent sentence whose circumscription is inconsistent, while the circumscription of a consistent universal sentence is always consistent [11]. If there is a need for some equality literals to exist in the given sentence, they can be simply expressed in another predicate symbol plus constrain axioms.

CONSIDER ONLY FREE MODELS. The objects which are beyond the free universe are not of user's interest, because they are unable to be identified by name.

Chapter 3

CP-resolution: A Proof-Theoretic Framework

In this chapter we introduce the CP-resolution to search for a sufficient condition under which a negative literal from P can be concluded with respect to a circumscriptive formula. The CP-resolution is based on the constructive negation and blocking formulas. Its purpose is to derive sequences of literals which are orthogonal to the definition of the negative literal from P, to implement the generalized predicate completion of that negative literal. Then, we introduce the *rule of completion*, based on the CP-resolution. Before we discuss the features of CP-resolution, we give intuition which leads to CP-resolution. We introduce *MIME-resolution* which is used to establish the definition of each negative literal from P with respect to circumscription. We explain that CP-resolution is orthogonal to the branches of a MIME-tree, and a CP-refutation implements the predicate completion of the definition of an atom $p(\boldsymbol{t}) \in pos(P)$ with respect to the semantics of circumscription.

3.1 Local Skolemization and Blocking Formulas

Let Π be a set of clauses. The *full program* of Π, denoted as $F(\Pi)$, is a set of chains defined as:

$$F(\Pi) = \{L_i \leftarrow \neg L_0, \ldots, \neg L_{i-1}, \neg L_{i+1}, \ldots, \neg L_n \mid \{L_0 \vee \cdots \vee L_n\} \in \Pi, 0 \leq i \leq n.\}.$$

Note that L_i is not an equality literal because no chain has an equality literals as its head.

A *factor* of clause C is $C\theta$, where θ is a most significant unifier of C. A set of clauses Π is said to be *closed under factorization* if Π contains every factor of

clauses in it. For convenience, we consider only full programs that are closed under factorization, as defined below.

Definition 3.1 Let Π be a set of clauses closed under factorization. The *generalized full program* of Π, denoted as $GF(\Pi)$, is a set of chains in the form:

$$C : L_0(\boldsymbol{x}) \leftarrow L_1, \ldots, L_m, \neg \cdot E,$$

where $L_0(\boldsymbol{x}) \leftarrow L_1, \ldots, L_m$ is the Clark's general form [8, 27] of a clauses in $F(\Pi)$, and $\neg \cdot E$ is a conjunction of inequalities $\neg e(\boldsymbol{s}, \boldsymbol{t})$ for each pair of literals $\rho(\boldsymbol{s})$ and $\neg\rho(\boldsymbol{t})$ in $\{\neg L_0, \ldots, L_m\}$. □

The process of adding $\neg \cdot E$ is called *strengthening*, which eliminates the trivial factors of the clauses such as $p(a) \leftarrow p(a)$ and $p(b) \leftarrow q(a), \neg q(a)$.

Example 3.1 Let Π contain the following clauses:

r1: $p(x) \vee \neg q(y, x)$;
r2: $p(x) \vee z(x)$;
r3: $p(a)$;
r4: $\neg z(b)$.

Then $GF(\Pi)$ is the set of the following chains:

C1: $p(x) \leftarrow q(y, x)$;
C2: $\neg q(y, x) \leftarrow \neg p(x)$;
C3: $p(x) \leftarrow \neg z(x)$;
C4: $z(x) \leftarrow \neg p(x)$;
C5: $p(x) \leftarrow e(x, a)$;
C6: $\neg z(x) \leftarrow e(x, b)$.

□

The variables in the top goal are *universal variables* (or uni-variables for short). The uni-variables in the top goal will be passed to the sequence of goals during the resolution. The variables in the head of a chain are also uni-variables in the sense that they will eventually be unified with the terms containing only variables passed from the top goal. An *existential variable* (or ex-variable for short) is a variable appearing in the body but not the head of a chain. As expected, the ex-variables will never be unified with any term, and will always exist in the sequence of goals same as when they are introduced. The ex-variables present a potential trouble to the completeness of computation and therefore should be eliminated through Skolemization.

Definition 3.2 Let Π be a set of clauses and P and Z be distinct sets of predicates in Π. Let the top goal contain the tuple of terms $\boldsymbol{t}$. Further, assume $C = L(\boldsymbol{x}) \leftarrow W(\boldsymbol{x}, y_1, \ldots, y_m)$, where y_i are ex-variables, is a chain in $GF(\Pi)$ such that $L \in pos(P) \cup any(Z)$. The *locally Skolemized chain* of C is a chain, obtained from C by Skolemizing each ex-variable in C against the tuple of terms in the top goal, as follows:

$$L(\boldsymbol{x}) \leftarrow W(\boldsymbol{x}, f_1(\boldsymbol{t}), \ldots, f_m(\boldsymbol{t})).$$

Like the ex-variables, a Skolem term will always exist in the sequence of goals same as when it is introduced. There does not exist any nested Skolem term.

The *locally Skolemized full program* of Π, denoted as $SF(\Pi)$, is then defined as the set of locally Skolemized chains of all chains in $GF(\Pi)$. □

Local Skolemization is named after the simple fact that Skolemization is applied to the body of each individual chain locally. The fact that the ex-variables are locally Skolemized against the tuple of terms in the top goal indicates that the local Skolemization is dynamically performed. The motivation of introducing the local Skolemization is to simulate a standard or global Skolemization.

Example 3.2 Let $GF(\Pi)$ be the set of chains as in Example 3.1. Let the top goal contain the tuple of terms $\boldsymbol{t}$. Then $SF(\Pi)$ contains the same chains as in $GF(\Pi)$ except C1:

C1: $p(x) \leftarrow q(f(\boldsymbol{t}), x)$;
C2: $\neg q(y, x) \leftarrow \neg p(x)$;
C3: $p(x) \leftarrow \neg z(x)$;
C4: $z(x) \leftarrow \neg p(x)$;
C5: $p(x) \leftarrow e(x, a)$;
C6: $\neg z(x) \leftarrow e(x, b)$.

□

Now we are in the position to define blocking formulas.

Definition 3.3 Let Π be a program, P and Z distinct sets of predicates in Π, and ρ a literal in $pos(P) \cup any(Z)$. Further, assume the following are all chains in $SF(\Pi)$ with ρ as the head:

$C1$: $\rho(\boldsymbol{x}) \leftarrow L_{1,1}, \ldots, L_{1,k_1}$,
$\ldots\ldots$,
Cq: $\rho(\boldsymbol{x}) \leftarrow L_{q,1}, \ldots, L_{q,k_q}$.

Then a *blocking formula* [50] of $\rho(\boldsymbol{x})$, is defined as a minimal chain:

$$\neg\rho(\boldsymbol{x}) \leftarrow \neg L_{1,s_1}, \ldots, \neg L_{q,s_q},$$

where $1 \leq s_q \leq k_q$, i.e., L_{j,s_j} is a literal from the body of C_j.

Π_{cp} is used to denote the set of all blocking formulas of Π. A blocking formula in Π_{cp} is called a *CP-chain*. □

Obviously, if there is no chain in $SF(\Pi)$ with $\rho(\boldsymbol{x})$ as the head then $\neg\rho(\boldsymbol{x}) \leftarrow TRUE$ is the only blocking formula with $\neg\rho(\boldsymbol{x})$ as the head. Similarly, if $\rho(\boldsymbol{x}) \leftarrow$ is contained in $SF(\Pi)$, then $\neg\rho(\boldsymbol{x}) \leftarrow FALSE$ is the only blocking formula of Π that has $\neg\rho(\boldsymbol{x})$ as the head.

Without lossing generality, we assume that block formulas are also ordered, not because it is necessary, but because of convenience.

The name of *blocking* comes from the fact that ρ could not be derived whenever $\neg L_{1,s_1} \wedge \cdots \wedge \neg L_{q,s_q}$ has been derived.

We say a CP-chain $\neg\rho(\boldsymbol{x}) \leftarrow \neg L_{1,s_1}, \ldots, \neg L_{q,s_q}$ is *orthogonal* to each chain $\rho(\boldsymbol{x}) \leftarrow L_{j,1}, \ldots, L_{j,k_j}$; or the sequence of literals from the body of a CP-chain $\neg L_{1,s_1}, \ldots, \neg L_{q,s_q}$ is orthogonal to the sequence of literals $L_{j,1}, \ldots, L_{j,k_j}$.

Let a CP-chain C be

$$C = \neg\rho(\boldsymbol{x}) \leftarrow \neg L_1, \ldots, \neg L_i, \ldots, \neg L_n.$$

For each literal $\neg L_i$, there must at least exist one chain $r = \rho(\boldsymbol{x}) \leftarrow D$ in $SF(\Pi\downarrow\rho)$ such that $L_i \in D$, and $L_j \notin D$ for any $j \neq i$. On the other hand, for each chain $r = \rho(\boldsymbol{x}) \leftarrow D$ in $SF(\Pi\downarrow\rho)$, there must at least exist one literal $\neg L_i$ in C such that $L_i \in D$.

Example 3.3 Let $SF(\Pi)$ contain the same locally Skolemized chains as in Example 3.2. Then Π_{cp} contains the following CP-chains:

$C1$: $\neg p(x) \leftarrow \neg q(f(\boldsymbol{t}), x), z(x), \neg e(x, a),$
$C2$: $z(x) \leftarrow \neg e(x, b),$
$C3$: $\neg z(x) \leftarrow p(x).$

There is no CP-chain with head $q(y, x)$ because $\neg q(y, x) \notin pos(P) \cup any(Z)$. □

Due to the fact that each chain in $SF(\Pi)$ is in Clark's general form, a CP-chain is correspondingly in Clark's general form. The variables contained in the body of each CP-chain must be either contained in its head if it appears in a non-Skolem term, or contained in the top goal otherwise.

Note, without the local Skolemization of each chain in $SF(\Pi)$, the generation of the blocking formulas might transform a universally quantified literal into an existentially quantified one, which would lead to an unsound result.

Using a Skolem term to replace an ex-variable is to assume that this Skolem term represents a tuple of objects in the universe which satisfy the existential variable. The mapping from the ground Skolem terms to the ground non-Skolem terms in the underlying free universe determines what objects the Skolem term really represents. At the final stage, namely in the computed answer, the Skolem terms will be replaced by the new existential variables as the following definition shows.

Definition 3.4 A formula Ψ' is an S-*generalization* of Ψ, denoted as $S_g(\Psi)$, if Ψ' is obtained from Ψ by replacing each Skolem term $g(\boldsymbol{t})$ with a new ex-variable $\boldsymbol{z}$ subscripted with $g(\boldsymbol{t})$, where $\boldsymbol{t}$ is the tuple of the uni-variables appearing in the top goal. □

Example 3.4 Let $\Psi = p(f(\boldsymbol{t}))$ where f is a Skolem function symbol. Then its S-generalization is $S_g(\Psi) = p(z_{f(\boldsymbol{t})})$. □

Note that the reason to subscript the variable is that $z_{f(\boldsymbol{t})}$ is a collection of variables instead of just one. The subscripted term is still subject to substitutions. For example let θ and θ' be ground substitutions such that $\boldsymbol{t}\theta = \boldsymbol{t}_1$ and $\boldsymbol{t}\theta' = \boldsymbol{t}_2$. Then $z_{f(\boldsymbol{t})}\theta = z_{f(\boldsymbol{t}_1)}$ while $z_{f(\boldsymbol{t})}\theta' = z_{f(\boldsymbol{t}_2)}$. With the subscript, it is easy to see that

$$S_g(\Psi\theta) = S_g(\Psi)\theta,$$

for non-Skolem substitution θ.

3.2 Rules of Inference

There is a set of *rules of inference*. One rule (*extension*) is the resolution with retention of one matching literal, made a framed literal to mark its "remaining on borrowed time". Another rule (*reduction*) simulates the "*ancestor resolution*". The third rule (*factorization*) performs factoring. The fourth rule (*completion*) simulates "predicate completion based on constructive negation", which will be discussed in Section 3.3. The fifth rule (*merging*) merges left the identical literals. The sixth rule (*tightening*) ensures that each instance of the goal will not be subsumed by its ancestors, if its equality formula is satisfiable. The last rule (*truncation*) removes the trailing framed literals which are no longer needed.

Each rule has one *parent goal.* Extension, simulating resolution, has also an *input chain* from a given set of chains, apart from the parent goal. Similarly, completion has a *CP-constructed chain* involved, together with the parent goal. Furthermore, we have to consider the range from which the literal is to be selected in the parent goal. Note, R is not a selection function. It only specifies the range of literals to select. We always select the rightmost literal which is in R.

Definition 3.5 Given a goal G as a parent goal and a variable disjoint chain C, where

$$G: \quad \leftarrow E_G, \omega_1, \ldots, \omega_m \qquad \textit{and} \qquad C: \quad \rho_0 \leftarrow E_C, \rho_1, \ldots, \rho_n,$$

and a range R of selection, if ω_m (selected) is the rightmost unframed literal in R, and is unifiable with ρ_0 using the *mgu* θ, then the extension rule yields the goal G_C, called R-extension goal of G and C, which is the goal

$$\leftarrow E_G, E_C\theta, \omega_1, \ldots, [\omega_m], \rho_1\theta, \ldots, \rho_n\theta$$

R can be omitted if it refers to the set of all literals. □

Note the unifier θ does not apply to $E_G, \omega_1, \ldots, \omega_m$ because C is in Clark's general form, and thus ρ_0 contains pure variable arguments.

Clearly, extension is just a modified resolution rule.

A chain in Clark's general form has its head containing only pure variables, while its body might contain equality formulas. The head is thus always unifiable with the selected literal which it shares the same predicate and sign, although the resulted equality formula may not necessarily be satisfiable with CET. Substitution has no effect to all the unselected literals.

Example 3.5 Let

$$G: \quad \leftarrow e(x_1, a), [\neg z_1(x_1, x_2)], p_2(x_2),$$

and

$$C: \quad p_2(y_1) \leftarrow e(y_2, f(y_1)), \neg z_1(c, y_2), z_2(y_1),$$

$p_2(y_1)$ and $p_2(x_2)$ can be unified with the *mgu* $\theta = \{y_1/x_2\}$. The $(pos(P))$-extension goal of G and C is the goal:

$$G_C: \quad \leftarrow e(x_1, a), e(y_2, f(x_2)), [\neg z_1(x_1, x_2)], [p_2(x_2)], \neg z_1(c, y_2), x_2(x_2).$$

□

Definition 3.6 Given a goal G as a parent goal where

$$G: \quad \leftarrow E_G, \omega_1, \ldots, \omega_m,$$

and a range R of selection, if $\omega_m(\boldsymbol{t}_m)$ (selected) is the rightmost unframed literal in R, and there exists a framed literal $[\omega_i(\boldsymbol{t}_i)]$, where $i < m$, such that ω_i shares the same predicate but different sign with ω_m, then the reduction rule may yield goal G_R, called the *R-reduction* goal of G, which is obtained from G by framing $\omega_m(\boldsymbol{t}_m)$ and then adding $e(\boldsymbol{t}_i, \boldsymbol{t}_m)$. R can be omitted if it refers to the set of all literals. □

As pointed out in [7], each reduction step corresponds to an ancestor resolution, and an ancestor resolution is needed in order to retain refutation completeness if and only if when a goal is reducible. In other words, the ancestor resolution whish is not corresponding to a reduction can be simply ignored.

Example 3.6 Let

$$G: \quad \leftarrow e(x_1, a), [\neg z_1(x_1, x_2)], p_2(x_2), z_1(x_1, b)$$

The $(any(Z))$-reduction goal of G is obtained from G by framing $z_1(x_1, b)$ and then adding $e(x_2, b)$ (since $e(x_1, x_1)$ is a tautology), and

$$G_R: \quad \leftarrow e(x_1, a), e(x_2, b), [\neg z_1(x_1, x_2)], p_2(x_2), [z_1(x, b)].$$

□

Definition 3.7 Given a goal G as a parent goal, where

$$G: \quad \leftarrow E_G, \omega_1, \ldots, \omega_m,$$

and a range R of the selection, if $\omega_m(\boldsymbol{t}_m)$ is the rightmost literal in R, and there exists another unframed literal $\omega_i(\boldsymbol{t}_i)$, where $i < m$, such that $\omega_i(\boldsymbol{t}_i)$ and $\omega_m(\boldsymbol{t}_m)$ are unifiable using the *mgu* θ, then the factorization rule may yield goal G_F, called *R-factoring* goal of G, which is obtained from $G\theta$ by adding the equality $e(\boldsymbol{t}_i, \boldsymbol{t}_m)$, and then deleting ω_m. □

Example 3.7 Let

$$G: \quad \leftarrow e(x_1, a), z_1(x_1, x_2), [p_2(x_2)], p_3(x_3), z_1(x_1, b).$$

The $(any(Z))$-factoring goal of G is obtained from G by adding $e(x_2, b)$ (since $e(x_1, x_1)$ is a tautology), and then deleting $z_1(x_1, b)$:

$$G_F: \quad \leftarrow e(x_1, a), e(x_2, b), z_1(x_1, x_2), [p_2(x_2)], p_3(x_3).$$

□

Definition 3.8 Given a goal G as a parent goal. The merging rule may yield goal G_M, called *merging* goal of G, which is obtained from G by merging left any identical unframed literals. □

Example 3.8 Let

$$G: \quad \leftarrow \neg q_1(x_1, b), [p_2(x_2)], \neg q_1(x_1, b), q_3(x_2),$$

The merging goal of G is obtained from G by merging left $q_1(x_1, b)$:

$$G_M: \quad \leftarrow \neg q_1(x_1, b), [p_2(x_2)], q_3(x_2).$$

□

Definition 3.9 Given a goal G as a parent goal, tightening rule may yield goal G_{TT}, called *tightening* goal of G, which is obtained from G by adding an inequality $\neg e(t_1, t_2)$ for each pair of literals $[\rho(t_1)]$ and $\rho(t_2)$, where $[\rho(t_1)]$ is a framed literal in the goal and $\rho(t_2)$ is an unframed literal which has no framed literal to its right. □

Example 3.9 Let the goal

$$G: \quad \leftarrow e(x_1, a), e(y_2, b), [\neg z_1(x_1, x_2)], [p_2(x_2)], \neg z_1(c, y_2).$$

The tightening goal of G_{TT} of G is obtained from G by adding the inequality $\neg e(\langle x_1, x_2\rangle, \langle c, y_2\rangle)$:

$$G_{TT}: \quad \leftarrow e(x_1, a), e(y_2, b), \neg e(\langle x_1, x_2\rangle, \langle c, y_2\rangle), [\neg z_1(x_1, x_2)], [p_2(x_2)], \neg z_1(c, y_2).$$

Note that $\neg e(\langle x_1, x_2\rangle, \langle c, y_2\rangle)$ is a 2-fold inequality and is equal to $\neg e(x_1, c) \vee \neg e(x_2, y_2)$. □

Definition 3.10 Given a goal G as a parent goal and a range R, truncation rule may yield goal G_T, called *R-truncation* goal of G, which is obtained from G by deleting the framed literals which are not followed by an unframed literals from R. Note R can be omitted if it refers to the set of all literals. □

Example 3.10 Let

$$G: \quad \leftarrow e(x_1, a), \neg p_1(x_1, x_2), [p_2(x_2)], q_3(x_1, b),$$

The $(pos(P))$-truncation goal of G is obtained from G by deleting $[p_2(x_2)]$ and

$$G_T: \quad \leftarrow e(x_1, a), \neg p_1(x_1, x_2), q_3(x_1, b).$$

□

To achieve a refutation completion for a first-order theory, only extension and reduction rules are needed [28]. Completion rule is needed for a nonmonotonic deduction. Factorization and merging rules are aimed at reducing the search space, merging is actually the special case of factorization with identity unifier. The purpose of tightening is also to reduce the search space by eliminating the instances of the goal which are subsumed by the goal's ancestor. The truncation rule discards the resolved literal which is no longer useful to the future deduction.

3.3 CP-resolution

We define CP-resolution in this subsection, and then give some examples to show how problems shown in Section 1.2.1 are overcome by CP-resolution.

Definition 3.11 Let S be a set of CP-chains, and $G = \leftarrow \neg p(t)$ the top goal, where $p \in pos(P)$. A *CP-derivation* of a goal G_n from $S \cup \{G\}$ is a sequence $G = G_0, G_1, \ldots, G_n$ of goals, such that

1. A goal G_{tmp} is the $(neg(P) \cup any(Z))$-extension goal of G_{i-1} and a chain C_i in S (*CP-extension*);
2. For each newly introduced literal $\rho(s_1)$ in G_{tmp}, if there exists a framed literal $[\rho(s_2)]$ in the goal, then either
   (a) replace $\rho(s_1)$ by the equality $e(s_1, s_2)$ (*CP-tightening*); or
   (b) leave G_{tmp} unchanged;
3. For each framed literal $[\omega(t_2)]$ in G_{tmp}, $\omega \in any(Z)$, if it shares the predicate symbol with $[\omega_q(t_1)]$ but has different sign, an inequality $\neg e(t_1, t_2)$ is added, where $[\omega_q(t_1)]$ is the last framed literal in the goal (*CP-reduction*);
4. Perform a $(neg(P) \cup any(Z))$-truncation;
5. The equality formula of G_i is satisfiable.

□

Definition 3.12 Let S be a set of chains, and $G = \leftarrow \neg p(\boldsymbol{t})$ the top goal, where $p \in pos(P)$. A goal G' is said to be a CP-leaf of $S \cup \{G\}$ if there exists a CP-derivation $G = G_0, G_1, \ldots, G_n = G'$, and G' contains no literals in $neg(P) \cup any(Z)$. A CP-derivation of $S \cup \{G\}$ which leads to a CP-leaf is called a *CP-refutation* of $S \cup \{G\}$. A chain $C = \neg p(\boldsymbol{t}) \leftarrow W$ is *CP-constructed* from S if the goal $\leftarrow W$ is a CP-leaf through a CP-refutation of $S \cup \{\leftarrow \neg p(\boldsymbol{t})\}$. □

Now we define completion reference rule which was not defined in Section 3.2.

Definition 3.13 Given a goal G as a parent goal and a CP-constructed chain C, or its variant, then the completion rule may yield the goal G_{CM}, called completion goal of G and C, which is the $(neg(P) \cup any(Q))$-extension goal of G and C. □

The purpose of CP-resolution of $S \cup \{\leftarrow \neg p(\boldsymbol{t})\}$ is to return a set of CP-constructed chains which constitute the predicate completion of the definition of $p(\boldsymbol{t})$ with respect to the semantics of CIR $(S(Q, P, Z); P)$. CP-resolution is thus a nonmonotonic operator.

Now let us give some examples to show how the problems listed in Section 1.2.1 can be overcome by using CP-resolution.

Example 3.11 This example is from ([27] p. 93), and is same as Example 1.1. Let Π contain:

$$p_1 \leftarrow \neg p_2(x); \quad p_2(x) \leftarrow e(x, a)$$

and the given goal is $\leftarrow p_1$. The derivation is:

G_0: $\leftarrow p_1$,
G_1: $\leftarrow \neg p_2(x)$,
at this point we switch to CP-resolution. And the only CP-chain we have is $\{\neg p_2(x) \leftarrow \neg e(x, a)\}$. The CP-refutation of $\Pi_{cp} \cup \{\leftarrow \neg p_2(x)\}$ is:
G'_0: $\leftarrow \neg p_2(x)$,
G'_1: $\leftarrow \neg e(x, a)$.
Therefore the chain $\neg p_2(x) \leftarrow \neg e(x, a)$ is the CP-constructed chain. The derivation can be carried on from G_1 with a completion reference rule:
G_2: $\leftarrow \neg e(x, a)$.

The answer to the query p_1 is $\neg e(x, a)$, which means that if there exists an object in the universe which is not a, then p_1 is true. Notice, since the variable x is not in the query, but is introduced during derivation, it is existentially quantified. □

Example 3.11 shows that we can proceed with the floundered query without worrying the unsoundness. The result will always be sound.

Example 3.12 Let Π contain the following clauses:

C1: $p(x,y) \vee \neg z(x) \vee \neg p(f(x),a)$,
C2: $z(b)$.

The relevant chains in $SF(\Pi)$ are:

C_1: $p(x,y) \leftarrow z(x), p(f(x),a)$,
C_2: $z(x) \leftarrow e(x,b)$.

In Section 3.4 we will see that the ordering in C_1 has a significant impact to the finiteness of MIME-tree. However, in this example we can see that CP-resolution overcomes the possible unfavorable ordering and returns a complete answer. The relevant CP-chains are:

C_1': $\neg p(x,y) \leftarrow \neg z(x)$;
C_2': $\neg p(x,y) \leftarrow \neg p(f(x),a)$; and
C_3': $\neg z(x) \leftarrow \neg e(x,b)$.

CP-refutation 1 for the give goal $\leftarrow \neg p(x,y)$:

G0: $\leftarrow \neg p(x,y)$;
G1: $\leftarrow [\neg p(x,y)], \neg z(x)$;
G2: $\leftarrow \neg e(x,b)$;

CP-refutation 2 for the give goal $\leftarrow \neg p(x,y)$:

G0: $\leftarrow \neg p(x,y)$;
G1: $\leftarrow [\neg p(x,y)], \neg p(f(x),a)$;
G2: $\leftarrow [\neg p(x,y)], [\neg p(f(x),a)], \neg z(f(x))$;
G3: $\leftarrow \neg e(f(x),b)$;

We still have CP-refutations 3, 4, $\cdots$, but let us right stop here, since the other CP-refutations are just redundant. The computed answer is $\forall x,y\ (\neg e(x,b) \vee \neg e(f(x),b) \vee \cdots)$. However, since $\forall(\neg e(f(x),b))$ is $TRUE$ under CET, the correct answer is $SEM \models \neg p(x,y) \leftrightarrow TRUE$. □

Example 3.13 Let Π contain the same chains as in Example 1.3, i.e., $\{p(x) \leftarrow q(x), p(f(x)); \neg q(x) \leftarrow e(x,a)\}$. The relevant CP-chains are $\{\neg p(x) \leftarrow \neg q(x); \neg p(x) \leftarrow \neg p(f(x))\}$. Note $\neg p(x) \leftarrow \neg q(x)$ is also a chain in Π° because it needs only one step of CP-derivation. The derivation is:

G_0: $\leftarrow \neg p(x)$;
G_1: $\leftarrow [\neg p(x)], \neg q(x)$;
G_2: $\leftarrow e(x, a)$.

Using CP-resolution can at least give the query $\neg p(x)$ an answer $e(x, a)$, which is certainly better than nothing at all. □

Example 3.13 indicates that, more than giving nothing-or-all answers, CP-resolution turns out answers whenever possible, even if the completion condition is not satisfied.

3.4 MIME-resolution, MIME-tree and the Weakly Bounded Term Size Property

The purpose for introducing MIME-resolution is to give insight int how CP-resolution is defined, the orthogonal relation between CP-resolution and MIME-resolution, and the weakly bounded term property. In this sense, MIME-resolution is only a concept we need to understand, if we ever want to understand how CP-resolution works. CP-resolution. We don't actually apply MIME-resolution.

There are several further refinements of the linear resolution, among them is ME-*resolution* [29], which Loveland himself called *model elimination procedure*. ME-resolution is essentially the same as OL-*resolution* [7], and therefore is a special version of SL-*resolution* [22, 7]. ME-resolution is both sound and refutation complete [29]. We assume that the readers are familiar with ME-resolution.

A goal is *acceptable* [29] if and only if it does not contain a pair of identical or complementary literals, either framed or not.

A goal is *admissible* [29] if and only if it satisfies the following:

(1) complementary unframed literals are not in the same cell;

(2) no unframed literal is to the right of an identical framed literal; and

(3) no framed literal is identical or complementary to another framed literal.

Basically, ME-resolution uses five rules of inference, which are reduction, factorization, extension, truncation and merging, and requires that each goal be acceptable.

In [29], Loveland also introduced a weaker version of ME-resolution, which we call WME-*resolution*. Intuitively, WME-resolution is ME-resolution without factorization and merging, and it tolerates tautologies as long as the complementary

unframed literals are not in the same cell. Instead of requiring the goal to be acceptable, WME-resolution requires the goal to be admissible. WME-resolution is sound and refutation complete too.

We will introduce *MInimal model ME-resolution*, or MIME-*resolution*. Intuitively, a MIME-resolution is a WME-resolution with ranges of selection imposed on inference rules.

Definition 3.14 Let S be a set of chains, and $G = \leftarrow p(\boldsymbol{t})$ the top goal, where $p \in pos(P)$. A *MIME-derivation* of a goal G_n from $S \cup \{G\}$ is a sequence $G = G_0, G_1, \ldots, G_n$ of goals, such that

1. for $1 \leq i \leq n$, G_i is either
   (a) a $(pos(P) \cup any(Z))$-extension goal of G_{i-1} and some chain C_i in S, followed by a tightening operation, or
   (b) a $any(Z)$-deduction goal of G_{i-1};
2. perform a $(pos(P) \cup any(Z))$-truncation;
3. the equality formula of G_i is satisfiable.

Further, a sequence of input chains $C_1, \ldots, C_n$ is called the input chains of the derivation, where C_i is empty if G_i is not derived by extension. □

Definition 3.15 Let S be a set of chains, and $G = \leftarrow p(\boldsymbol{t})$ the top goal, where $p \in pos(P)$. The *MIME-tree* of $S \cup \{G\}$ is the search space of all the possible MIME-derivations of $S \cup \{G\}$. □

Definition 3.16 Let S be a set of chains, and $G = \leftarrow p(\boldsymbol{t})$ the top goal, where $p \in pos(P)$. A goal G' is said to be a MIME-leaf of $S \cup \{G\}$ if there exists a MIME-derivation $G = G_0, G_1, \ldots, G_n = G'$ and G' contains no literals in $pos(P) \cup any(Z)$. A MIME-derivation of $S \cup \{G\}$ which leads to a MIME-leaf is called a *MIME-refutation* of $S \cup \{G\}$. □

Example 3.14 Let S contain the following chains:

C1: $p(x) \leftarrow z_1(x)$,
C2: $z_1(x) \leftarrow z_2(x), q(a)$,
C3: $z_2(x) \leftarrow \neg p(b)$.

The following is a MIME-derivation of $S \cup \{\leftarrow p(x)\}$:

G:	$\leftarrow p(x)$;	
G_1:	$\leftarrow [p(x)], z_1(x)$;	extension using C1
G_2:	$\leftarrow [p(x)], [z_1(x)], z_2(x), q(a)$;	extension using C2
G_3:	$\leftarrow [p(x)], [z_1(x)], [z_2(x)], q(a), \neg p(b)$;	extension using C3
G_4:	$\leftarrow e(x,b), q(a)$;	reduction followed by truncation.

Note that G_4 is a MIME-leaf of $S \cup \{\leftarrow p(x)\}$, and the MIME-derivation is a MIME-refutation. □

In Example 3.14, if S contains only C1 and C2, then the MIME-derivation of $S \cup \{\leftarrow p(x)\}$ would fail at G_3. In that case, G_3 is not a MIME-leaf, and the MIME-derivation is not a MIME-refutation.

Definition 3.17

1. A set of clauses Π is said to possesses *bounded term size* property [45] if and only if every MIME-tree of $GF(\Pi) \cup \{\leftarrow p(t)\}$ is finite, for every $p(t) \in pos(P)$ in Π.
2. A set of clauses Π is said to possesses the *weakly bounded term size* property if there exists a *single* finite MIME-tree of $GF(\Pi) \cup \{\leftarrow p(t)\}$, for every $p(t) \in pos(P)$ in Π.

□

The difference between the bounded term size and the weakly bounded term size properties is the former requires all the resolution trees to be finite, while the latter requires only one. The latter is thus a weaker completion condition than the former.

Note, a MIME-tree of $GF(\Pi) \cup \{\leftarrow p(t)\}$ could be finite or infinite, depending on the ordering of the literals in each chain in $GF(\Pi)$. Example 3.15 shows this, and thus proves that there exists set of clauses which possesses the weakly bounded term size property, but not bounded term size property.

Example 3.15 Let Π contain the following clauses (same as Example 3.12):

C1: $p(x,y) \vee \neg z(x) \vee \neg p(f(x),a)$,
C2: $z(b)$.

The relevant chains in $GF(\Pi)$ are:

C_1: $p(x,y) \leftarrow z(x), p(f(x),a),$
C_1': $p(x,y) \leftarrow p(f(x),a), z(x),$ in another ordering
C_2: $z(x) \leftarrow e(x,b).$

The MIME-derivation of $GF(\Pi) \cup \{\leftarrow p(x,y)\}$ in the first ordering is:

G_0: $\leftarrow p(x,y);$
G_1: $\leftarrow [p(x,y)], z(x), p(f(x),a);$
G_2: $\leftarrow [p(x,y)], z(x), [p(f(x),a)], z(f(x)), p(ff(x),a);$
.....

Apparently, there is an infinite branch in the MIME-tree of $GF(\Pi) \cup \{\leftarrow p(x,y)\}$. The MIME-derivation of $GF(\Pi) \cup \{\leftarrow p(x,y)\}$ in the second ordering is:

G_0': $\leftarrow p(x,y);$
G_1': $\leftarrow [p(x,y)], p(f(x),a), z(x);$
G_2': $\leftarrow e(x,b), [p(x,y)], p(f(x),a);$
G_3': $\leftarrow e(x,b), [p(x,y)], [p(f(x),a)], p(ff(x),a), z(f(x));$
G_4': $\leftarrow e(x,b), e(f(x),b), [p(x,y)], [p(f(x),a)], p(ff(x),a).$

G_4' fails since $e(x,b) \wedge e(f(x),b)$ is not satisfiable under CET. The MIME-tree of $GF(\Pi) \cup \{\leftarrow p(x,y)\}$ is finite. □

Recall Example 3.12 in which Π contains the same chains as in that of Example 3.15, and which returns a complete answer. This indicates that as long as there is a finite MIME-tree, the CP-resolution is complete. The reason why CP-resolution is insensitive to the ordering of the literals in each chain lies in the trick of blocking formulas, which are not sensitive to the ordering.

The following *Lifting Lemma* is very useful in proofs to lift the ground cases into general cases.

Lemma 3.1 (Lifting Lemma) *Let G be a goal and C be a chain. Let $G' = G\theta$ be an instance of G, where θ is a non-Skolem substitution, and C' be an instance of C. If the extension rule yields a goal $G'_{C'}$ of G' and C', then the extension rule must also yield a goal G_C of G and C, and there further exist non-Skolem substitutions λ and θ' such that $G'_{C'} = G_C\lambda$ and $\lambda = \theta\theta'$.* □

Proof: Let

$$G: \leftarrow \rho_1, \ldots, \rho_m \quad \text{and} \quad C: \omega_0 \leftarrow \omega_1, \ldots, \omega_n.$$

Assume $C' = C\sigma$, and $\rho_m\theta = \omega_0\sigma$. then $G'_{C'}$ is the goal

$$G'_{C'}: \quad \leftarrow \rho_1\theta, \ldots, [\rho_m\theta], \omega_1\sigma, \ldots, \omega_n\sigma;$$

Assume γ is the substitute such that $\rho_m = \omega_0\gamma$. Notice that the domain of γ is the variables in ω_0. Then G_C is the goal

$$G_C: \quad \leftarrow \rho_1, \ldots, [\rho_m], \omega_1\gamma, \ldots, \omega_n\gamma;$$

$\rho_m\theta = \omega_0\gamma\theta$ holds due to $\rho_m = \omega_0\gamma$. The body of C either contain some Skolem terms or some variables not in ω_0, but not both. In either case, there exists a non-Skolem substitution θ', possibly identity substitution, such that $\omega_i\gamma\theta\theta' = \omega_i\sigma$ for $1 \leq i \leq n$. Notice $\rho_i\theta\theta' = \rho_i\theta$ for $1 \leq i \leq n$, because the domain of θ' is disjoint to the variables in $G\theta$. Therefore, $G_C\theta\theta' = G'_{C'}$. This completes the proof. □

Since reduction is a special case of extension, the result of Lifting Lemma can be easily extended to whole ME-resolution.

3.5 Orthogonality of CP-resolution and MIME-resolution

CP-derivation is orthogonal to MIME-resolution. Every detail of CP-resolution is defined exactly corresponding to MIME-resolution.

Definition 3.18 A *cut* in a MIME-tree is a set of nodes which contains exactly one node from each branch. A *perfect cut* consists of all leaf nodes of the MIME-tree. A *blocking formula* of a cut $\Xi = \{N_1, \ldots, N_m\} = \{\leftarrow W_1, \ldots, \leftarrow W_m\}$ is $B = \{\neg\omega_1, \ldots, \neg\omega_m\}$ such that $\omega_i \in W_i$, for $1 \leq i \leq m$. Each node N_i is *blocked* by B through $\neg\omega_i$. A node N_i is *consolidatedly blocked* by B if there is a literal $\omega \in neg(P) \cup any(Q)$ is in N_i and $\neg\omega$ is in B. The blocking formula is *consolidated* if $\neg\omega_i \in pos(P) \cup any(Q)$, for $1 \leq i \leq m$. □

If we view B as a set, then each element in it may block one or more nodes in the cut Ξ, the number of elements in B is less than or equal to the number of nodes in Ξ.

Definition 3.19 A cut in a MIME-tree is measured with a *level* number with regard to its blocking formula. The cut containing root node only is of level 1. A cut Ξ_2 (level k), which is said to be one level lower than the cut Ξ_1 (level $k-1$), is obtained from Ξ_1 by replacing a nonleaf node with all its children nodes. □

Note a level 1 cut could not be consolidatedly blocked. The replaced node could not be consolidatedly blocked either.

Theorem 3.2 *A goal in a CP-derivation of* $\Pi_{cp} \cup \{\leftarrow \neg p(\boldsymbol{t})\}$ *is always a bocking formula of a cut in MIME-tree of* $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$. *Furthermore, if the goal* G_{i-1} *is a blocking formula of a cut* Ξ_1 *in the MIME-tree* $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$, *then the goal* G_i *is a blocking formula of a cut* Ξ_2, *which is one level lower than* Ξ_1. □

Proof: See Appendix. □

Due to the fact that MIME-resolution does not select the literals in $neg(P) \cup any(Q)$ of a node N in a MIME-tree, those literals will be passed to all the descendants of N all the way down to the leaf nodes. Let Ξ_1 be a cut in a MIME-tree, and G its blocking formula, which consolidatedly blocks Ξ_1. In case the MIME-tree is finite, then the cut Ξ_1 is equivalent to the cut Ξ_2, which is obtained from Ξ_1 by replacing each node N_i in Ξ_1 with all its leaf descendents. It is thus easy to see that from the level 1 cut, which contains the root node only, if we keep going one level lower at a time, we will reach the perfect cut in finite steps if the MIME-tree is finite.

Example 3.16 Let a node G_i in a MIME-tree be:

$$G_i = \leftarrow [\rho_0(\boldsymbol{t}_0)], \ldots, [\rho_j(\boldsymbol{t}_1)], \ldots, [\rho_{s-1}], \neg\rho_j(\boldsymbol{t}_2),$$

and let G_i' be a goal of CP-derivation of $\Pi_{cp} \cup \{G_0'\}$ which is a blocking formula of a cut Ξ in a MIME-tree of $SF(\Pi) \cup \{G_0\}$:

$$G_i' = \leftarrow [\neg\rho_0(\boldsymbol{t}_0)], \ldots, [\neg\rho_j(\boldsymbol{t}_1)], \ldots, [\neg\rho_{s-1}], \rho_j(\boldsymbol{t}_2).$$

Let $\theta = \{\boldsymbol{x}/\boldsymbol{t}_2\}$. Let C_1 and C_2 be all the chains in $SF(\Pi)$ whose head is $\neg\rho_j(\boldsymbol{x})$, and

$C_1\theta$: $\neg\rho_j(\boldsymbol{t}_2) \leftarrow \omega_{1,1}, \rho_0(\boldsymbol{t}_3)$;
$C_2\theta$: $\neg\rho_j(\boldsymbol{t}_2) \leftarrow \omega_{2,1}, \omega_{2,2}$;

The relevant CP-chains in Π_{cp} after applying θ are:

$C_1'\theta$: $\rho_j(\boldsymbol{t}_2) \leftarrow \neg\omega_{1,1}, \neg\omega_{2,1}$;
$C_2'\theta$: $\rho_j(\boldsymbol{t}_2) \leftarrow \neg\omega_{1,1}, \neg\omega_{2,2}$;
$C_3'\theta$: $\rho_j(\boldsymbol{t}_2) \leftarrow \neg\rho_0(\boldsymbol{t}_3), \neg\omega_{2,1}$;
$C_4'\theta$: $\rho_j(\boldsymbol{t}_2) \leftarrow \neg\rho_0(\boldsymbol{t}_3), \neg\omega_{2,2}$;

All the children of G_i in the MIME-tree (before the $(pos(P) \cup any(Z))$-truncation is performed) are:

$$G_{i+1,1} = \leftarrow [\rho_0(t_0)], \ldots, [\rho_j(t_1)], \ldots, [\rho_{s-1}], [\neg\rho_j(t_2)], e(t_1, t_2);$$
$$G_{i+1,2} = \leftarrow [\rho_0(t_0)], \ldots, [\rho_j(t_1)], \ldots, [\rho_{s-1}], [\neg\rho_j(t_2)], \omega_{1,1}, \rho_0(t_3), \neg e(t_0, t_3);$$
$$G_{i+1,3} = \leftarrow [\rho_0(t_0)], \ldots, [\rho_j(t_1)], \ldots, [\rho_{s-1}], [\neg\rho_j(t_2)], \omega_{2,1}, \omega_{2,2};$$

The possible G'_{i+1} are:

$$G'_{i+1,1} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], \neg\omega_{1,1}, \neg\omega_{2,1}, \neg e(t_1, t_2);$$
$$G'_{i+1,2} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], \neg\omega_{1,1}, \neg\omega_{2,2}, \neg e(t_1, t_2);$$
$$G'_{i+1,3} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], \neg\rho_0(t_3), \neg\omega_{2,1}, \neg e(t_1, t_2);$$
$$G'_{i+1,4} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], \neg\rho_0(t_3), \neg\omega_{2,2}, \neg e(t_1, t_2);$$
$$G'_{i+1,5} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], e(t_0, t_3), \neg\omega_{2,1}, \neg e(t_1, t_2);$$
$$G'_{i+1,6} = \leftarrow [\neg\rho_0(t_0)], \ldots, [\neg\rho_j(t_1)], \ldots, [\neg\rho_{s-1}], [\rho_j(t_2)], e(t_0, t_3), \neg\omega_{2,2}, \neg e(t_1, t_2);$$

where $\neg e(t_1, t_2)$ in each goal is obtained through CP-reduction; $e(t_0, t_3)$ in $G'_{i+1,5}$ and $G'_{i+1,6}$ are obtained through CP-tightening.

$G'_{i+1,1}$ through $G'_{i+1,6}$ are all blocking formulas of the cut Ξ' of the MIME-tree which is obtained from Ξ by replacing G_i with all its children. Moreover, they have exhausted all the possible combinations to block all the children of G_i. □

Theorem 3.2 states that given a MIME-tree of $SF(\Pi) \cup \{\leftarrow p(t)\}$ and a goal G_i in a CP-derivation of $\Pi_{cp} \cup \{\leftarrow \neg p(t)\}$, we can always find a cut Ξ in the MIME-tree such that G_i is a blocking formula of Ξ. The reverse is not always true. Let B be a blocking formula of a cut Ξ in a MIME-tree of $SF(\Pi) \cup \{\leftarrow p(t)\}$, there does not necessarily exist a goal G_i in a CP-derivation of $\Pi_{cp} \cup \{\leftarrow \neg p(t)\}$ such that G_i is equal to B. This is because we only select the rightmost literal in $neg(P) \cup any(Z)$ in CP-derivation. We shall further discuss this issue in Section 3.7, when we prove the completeness of CP-resolution.

3.6 Procedural Definition of Circumscription

In this section, we define circumscription procedurally in terms of MIME-resolution, which gives the insight why CP-resolution is the building block to compute circumscription.

Definition 3.20 Let F be a set of chains and $G = \leftarrow p(\boldsymbol{t})$ the top goal, where $p \in pos(P)$. Then the *MIME-precondition* of $F \cup \{G\}$, denoted as $Prec(F, G)$, is defined as the formula

$$Prec(F, G) \equiv \bigvee_{W_i \in W_{F,G}} \exists_Y W_i(X, Y), \tag{3.1}$$

where $W_{F,G} = \{W_i | \leftarrow W_i$ *is a MIME-leaf of* $F \cup \{G\}\}$, X is the tuple of variables in G, while the existential quantifier quantifies the tuple Y of variables that are not in G. □

Intuitively, $Prec(F, G)$ is a disjunction of the bodies of every MIME-leave on the MIME-tree of $F \cup \{G\}$.

By definition, $F \cup Prec(F, \{\leftarrow p(\boldsymbol{t})\}) \models p(\boldsymbol{t})$ holds, i.e., $Prec(F, \{\leftarrow p(\boldsymbol{t})\})$ is the precondition that $p(\boldsymbol{t})$ can be derived from F. We show that for each D such that $D \in neg(P) \cup any(Q)$ and $F \cup D \models p(\boldsymbol{t})$, D implies $Prec(F, \{\leftarrow p(\boldsymbol{t})\})$.

Lemma 3.3 *Let D_i be a finite conjunction of literals such that $D_i \in neg(P) \cup any(Q)$ and*

$$\Pi \models p(\boldsymbol{a}) \leftarrow D_i \qquad \text{and} \qquad \Pi \not\models \neg D_i,$$

where $p(\boldsymbol{a}) \in pos(P)$. Then

$$\Pi \cup D_i \models \bigvee_j W_j.$$

Each W_j corresponds to a ground instance $\leftarrow W_j$ of a MIME-leaf of $GF(\Pi) \cup \{\leftarrow p(\boldsymbol{a})\}$. □

Proof: See Appendix. □

By Lemma 3.3, for every conjunction of literals $D_i \in neg(P) \cup any(Q)$, and $\Pi \not\models \neg D_i$, $\Pi \cup D_i$ implies $Prec(\Pi, \{\leftarrow p(\boldsymbol{t})\})$. Therefore $Prec(\Pi, \{\leftarrow p(\boldsymbol{t})\})$ could be considered as a theory closed under such D_i, for $1 \leq i \leq \infty$.

Due to the restriction on the range of selected literals, i.e., literals in $any(Q)$ are not selected in the extension rule in MIME-resolution, for an D_i stated above, we don't expect that there is a W_j in $Prec(\Pi, \{\leftarrow p(\boldsymbol{t})\})$ such that $\Pi \cup D_i \models W_j$ holds, as illustrated in the following example.

Example 3.17 Consider Π defined by

$$p \vee \neg z_1; \quad p \vee \neg z_2 \vee \neg q_1; \quad z_1 \vee \neg q_1 \vee \neg q_2; \quad z_2 \vee q_2.$$

Then $\Pi \wedge q_1 \models p$, but $\leftarrow q_1$ is not an MIME-leaf of $\Pi \cup \{\leftarrow p\}$. Instead, there exist two MIME-leaves of $\Pi \cup \{\leftarrow p\}$:

$$\leftarrow q_1, q_2 \quad \textit{and} \quad \leftarrow \neg q_2, q_1.$$

Obviously, we have $q_1 \equiv (q_1 \wedge q_2) \vee (q_1 \wedge \neg q_2)$, and Lemma 3.3 holds. □

Theorem 3.4 *Let* Π *be a consistent, universal sentence without equalities and* $\Pi^*(Q, P, Z)\downarrow p(\boldsymbol{t})$ *and* $\Pi^*(Q, P, Z)$ *be respectively defined as:*

$$\Pi^*(Q, P, Z)\downarrow p(\boldsymbol{t}) = \forall \boldsymbol{x}(p(\boldsymbol{t}) \rightarrow Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})).$$

and

$$\Pi^*(Q, P, Z) = \{\Pi^*(Q, P, Z)\downarrow p(\boldsymbol{t}) \mid \text{ for all } p(\boldsymbol{t}),\ p(\boldsymbol{t}) \in pos(P)\}.$$

Then

$$CIR(\Pi; P; Z) \equiv \Pi \cup \Pi^*. \tag{3.2}$$

□

Proof: By Theorem 2.7 and Lemma 3.3 we have proved the ground case. By Lifting Lemma, the general case is also true. □

By Theorem 3.4, together with the fact $\Pi \models \forall \boldsymbol{x}(p(\boldsymbol{t}) \leftarrow Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\}))$ the following holds:

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}\ (p(\boldsymbol{t}) \leftrightarrow Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})).$$

Therefore, $Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ is the *definition* of $p(\boldsymbol{t})$ with respect to the semantics of circumscription, or *definition* of $p(\boldsymbol{t})$ for short. $\forall \boldsymbol{x}\ (p(\boldsymbol{t}) \leftarrow Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ is the "*if*" part of the definition of $p(\boldsymbol{t})$; while $\forall \boldsymbol{x}\ (p(\boldsymbol{t}) \rightarrow Prec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ is the "*only if*" part of the definition of $p(\boldsymbol{t})$, which is in fact a *predicate completion* with respect to the semantics of circumscription.

Note Π^* is actually *procedural circumscription axiom.* It transforms a syntactic second-order formula into an equivalent proof procedural one.

The very existence of existential quantifier in (3.2) (refer to Equation 3.1) restricts applications in developing algorithms for computing circumscription from Π^*. At first thought, Skolemization may help to transform Π^* into a universal sentence. However, Skolemization can not be performed until $\Pi^*\downarrow p(\boldsymbol{t})$ is completely computed, which demands that the MIME-tree of $GF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$ to be finite.

This can be solved by local Skolemization. Instead of Skolemize Π^*, we locally Skolemize each chain in $GF(\Pi)$, i.e., instead of using $GF(\Pi)$, we use the locally Skolemized full program $SF(\Pi)$ of Π. This totally eliminates the existential variables, since every variable contained in the body of a chain is either contained in the top goal, if it appears in the some Skolem term, or contained in the head otherwise. The variables appearing in the head of each chain will eventually be unified with some terms containing only the uni-variables in the top goal.

Definition 3.21 Let Π be a consistent, universal sentence without equalities and $\Pi^\circ(Q,P,Z)\downarrow p(\boldsymbol{t})$ and $\Pi^\circ(Q,P,Z)$ be defined as

$$\Pi^\circ(Q,P,Z)\downarrow p(\boldsymbol{t}) = \forall \boldsymbol{x}(p(\boldsymbol{t}) \rightarrow Prec(SF(\Pi), \{\leftarrow p(\boldsymbol{t})\}).$$

where $SF(\Pi)$ is the locally Skolemized full program of Π, and

$$\Pi^\circ(Q,P,Z) = \{\Pi^\circ(Q,P,Z)\downarrow p(\boldsymbol{t}) \mid \textit{ for all } p(\boldsymbol{t}),\ p(\boldsymbol{t}) \in pos(P)\}.$$

Π° and $\Pi^\circ\downarrow p(\boldsymbol{t})$ are called *local Skolemization* of Π^* and $\Pi^*\downarrow p(\boldsymbol{t})$, respectively. □

Obviously, Π° is a universal sentence, therefore conceptually it is a set of clauses. $Prec(SF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ contains only the variables appearing in $p(\boldsymbol{t})$, and the existential quantifier is thus dropped (refer to Equation 3.1). The MIME-tree of $SF(\Pi)\cup\{\leftarrow p(\boldsymbol{t})\}$ might still be infinite, but now we don't have to wait until the tree is complete and then Skolemize it, since the Skolemization is already done in each chain in $SF(\Pi)$. The CP-resolution can now be defined orthogonal to the MIME-tree without worrying of unsoundness, so that chains in Π° can be CP-constructed through CP-refutation.

Definition 3.22 Let Φ be a formula with Skolem functions and Ψ a sentence. Then we say $\Phi \models_S \Psi$ if and only if

$$\Phi \cup E_S \models \Psi,$$

for every E_S, where E_S represents a second-order formula which states that for each ground Skolem term α, there exists a ground non-Skolem term a such that $e(\alpha, a)$ holds. □

The intuition of definition 3.22 is that the Skolem function symbols do not belong to the underlying language, and introducing of Skolem functions does not, and should not, change the underlying universe, so that the original issue can be investigated in the same environment, with Skolemization as analysis tool. Note, unique name axioms (CET6, CET7 and CET8) have no effect with Skolem functions.

Theorem 3.5 *Let Φ be a sentence, and $sklm(\Phi)$ be its Skolem normal form, then Φ is satisfiable if and only if $sklm(\Phi) \wedge \bigvee E_S$ is satisfiable.* □

Proof: The proof of Proposition 2.3 based on Proposition 2.2 also proves Theorem 3.5, i.e., Φ has a free model M, if and only if $sklm(\Phi)$ has a free model M' which is an extension of M with all ground Skolem terms mapping to non-Skolem ground terms. □

Remark 3.6 One fundermental requirement with Skolemization is the uniqueness of the Skolem functions, i.e., different ex-variable should be replaced by different Skolem function during Skolemization. The extension rule requires that the given chain is variable disjoint with respect to the given goal. Therefore in a MIME-derivation of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$, every time a locally Skolemized chain from $SF(\Pi)$ is used as an input chain, it should have unique Skolem function. This includes the situation a same locally Skolemized chain being used twice in a MIME-derivation.

On different occasions, if the blocking formulas with the same head need to be generated independently, they should be generated from the chains which have been Skolemized with different Skolem functions. Otherwise the uniqueness of the Skolem function is violated. □

We denote the Skolem normal form of $Prec(\Pi, \{\leftarrow \quad p(\boldsymbol{t})\})$ as $SPrec(\Pi, \{\leftarrow p(\boldsymbol{t})\})$. And $sklm(\Pi^*{\downarrow}p(\boldsymbol{t}))$ and $sklm(\Pi^*)$ are defined as

$$sklm(\Pi^*{\downarrow}p(\boldsymbol{t})) = \forall \boldsymbol{x}(p(\boldsymbol{t}) \rightarrow SPrec(GF(\Pi), \{\leftarrow p(\boldsymbol{t})\})).$$

and

$$sklm(\Pi^*) = \{sklm(\Pi^*{\downarrow}p(\boldsymbol{t})) \mid \quad for\ all\ \ p(\boldsymbol{t}), p(\boldsymbol{t}) \in pos(P)\}.$$

Due to the uniqueness of the Skolem function symbol, wherever there is a unique n-place Skolem function in $sklm(\Pi^*)$, if and only if there is a unique n-place Skolem function at the corresponding place in Π°. Therefore, the difference between $sklm(\Pi^*)$ and Π° is just a matter of Skolem function renaming.

Now, the two main theorems for this section.

Theorem 3.7 *Let Ψ be a quantifier free formula without Skolem function symbols. Then, assuming* CET,

$$CIR(\Pi; P; Z) \models \forall(\Psi) \quad \text{if and only if} \quad \Pi \cup \Pi^\circ \models_S \forall(\Psi). \tag{3.3}$$

□

Proof: By Theorem 3.4, $CIR(\Pi; P; Z) \equiv \Pi \cup \Pi^*$; by Theorem 3.5, $\Pi \cup \Pi^* \models \forall(\Psi)$ if and only if $\Pi \cup sklm(\Pi^*) \models_S \forall(\Psi)$; further more, Π° is just a Skolem function renaming of $sklm(\Pi^*)$, thus the theorem follows. □

Theorem 3.8 *Let Ψ be a quantifier free formula. Then, assuming CET,*

$$\Pi \cup \Pi^\circ \models_S \forall(\Psi) \quad \text{only if} \quad CIR(\Pi; P; Z) \models \forall(\exists S_g(\Psi)), \tag{3.4}$$

where Ψ contains only the Skolem functions in Π°, $S_g(\Psi)$ is the S-generalization of Ψ, and $\exists$ quantifies the variable introduced during S-generalization. □

Proof: We prove $\Pi \cup \Pi^\circ \models_S \Psi\theta$ only if $CIR(\Pi; P; Z) \models \exists \boldsymbol{y} S_g(\Psi)\theta$ for arbitrary ground non-Skolem substitution θ, which substitutes tuples of uni-variables by ground non-Skolem terms. By Theorem 3.4, $CIR(\Pi; P; Z) \equiv \Pi \cup \Pi^*$; by the fact that Π° is just a Skolem function renaming of $sklm(\Pi^*)$, we need only prove that $\Pi \cup sklm(\Pi^*) \models_S \Psi'\theta$ only if $\Pi \cup \Pi^* \models_S \exists \boldsymbol{y} S_g(\Psi')\theta$, where Ψ' is obtained from Ψ by Skolem function renaming same way as $sklm(\Pi^*)$ is renamed from Π°. Note $\Pi \cup \Pi^*$ contains no Skolem function, therefore $\Pi \cup \Pi^* \models_S \exists \boldsymbol{y} S_g(\Psi)\theta$ if and only if $\Pi \cup \Pi^* \models_S \exists \boldsymbol{y} S_g(\Psi')\theta$. By Theorem 3.5, $\Pi \cup \Pi^* \models \exists \boldsymbol{y} S_g(\Psi)\theta$ if and only if $\Pi \cup sklm(\Pi^*) \models_S \exists \boldsymbol{y} S_g(\Psi)\theta$. Now we need only prove that $\Pi \cup sklm(\Pi^*) \models_S \Psi'\theta$ only if $\Pi \cup sklm(\Pi^*) \models_S \exists \boldsymbol{y} S_g(\Psi')\theta$.

Assume $\Pi \cup sklm(\Pi^*) \models_S \Psi'\theta$, but $\Pi \cup sklm(\Pi^*) \not\models_S \exists \boldsymbol{y} S_g(\Psi')\theta$, or equivalently $\Pi \cup sklm(\Pi^*) \cup \forall \boldsymbol{y} \neg S_g(\Psi')\theta \cup \bigvee E_S$ is satisfiable. Let M be the model of $\Pi \cup sklm(\Pi^*) \cup \forall \boldsymbol{y} \neg S_g(\Psi')\theta \cup \bigvee E_S$. There must exist an E_S such that $M \models E_S$. There must also exist ground substitution σ which substitutes the tuple of variables $\boldsymbol{y}$ with ground non-Skolem terms in the way that $E_S \models S_g(\Psi'\theta)\sigma \leftrightarrow \Psi'\theta$. Note, $S_g(\Psi'\theta) = S_g(\Psi')\theta$ for non-Skolem substitution θ. It is obvious that $M \models \Pi \cup sklm(\Pi^*) \cup \neg S_g(\Psi')\theta\sigma \cup \bigvee E_S$. Therefore $M \models \Pi \cup sklm(\Pi^*) \cup \neg \Psi'\theta \cup \bigvee E_S$. This contradicts the assumption that $\Pi \cup sklm(\Pi^*) \models_S \Psi'\theta$. This completes the proof of Theorem 3.8. □

Note Π° is actually *locally Skolemized procedural circumscription axiom.* In general, it is not equivalent to Π^*, but is a simulation of Π^* instead.

Theorem 3.7 and 3.8 convert a semantic issue to a procedural one. Π° can be used to simulate Π^*, and Π° is a universal sentence and thus is equivalent to a set of clauses. Due to the semidecidability, we can not expect the MIME-derivation trees to be always finite. As a result, we might not be able to obtain every clause in Π°. CP-resolution is used to construct a chain C in Π°, or to construct a chain C which is subsumed by a chain C' in Π°. If the given theory possesses the weakly bounded term size property, then CP-resolution can construct the chain C for every C in Π°.

3.7 Property Analysis

This section is to give soundness and completeness of CP-resolution.

There are two kinds of leaf nodes on a MIME-tree, one is MIME-leaves, and the other is failed leaf nodes, i.e., the ones whose equality formulas are not satisfiable. There does not exist a goal whose selected literal can not be unified with the head of any chain in $SF(\Pi)$, because the chains in $SF(\Pi)$ are in Clark's generalized form with head to be pure variable literal. There might be some infinite branches. But to simplify our analysis, we assume without loss of generality they fail at ω, the *first limit ordinal.*

Recall that $Prec(\Pi^\circ, \{\leftarrow p(\boldsymbol{t})\})$ is defined as $Prec(\Pi^\circ, \{\leftarrow p(\boldsymbol{t})\}) \equiv \bigvee_i W_i$, for all W_i such that $\leftarrow W_i$ is a MIME-leaf of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$. It does not change the definition of $Prec(\Pi^\circ, \{\leftarrow p(\boldsymbol{t})\})$ if we expand it to contain all W_i such that $\leftarrow W_i$ is a leaf in the MIME-tree of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$, since the W_i corresponding to a failed leaf is *FALSE* with respect to CET.

Theorem 3.9 (Soundness of CP-resolution) *Let $G = \leftarrow W$ be a CP-leaf of a CP-derivation of $\Pi_{cp} \cup \{\leftarrow \neg p(\boldsymbol{t})\}$. Then $\Pi^\circ \models_S \forall \boldsymbol{x}(W \rightarrow \neg p(\boldsymbol{t}))$, where $\boldsymbol{x}$ is a tuple of variables in $\boldsymbol{x}$.* □

Proof: We prove $\Pi^\circ \models \forall \boldsymbol{x}(W \rightarrow \neg p(\boldsymbol{t}))$ by proving that each CP-leaf of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$ is a consolidated blocking formula of the cut which consists of all the leaf nodes of the MIME-tree of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$. By Theorem 3.2, a CP-goal is a blocking formula of a cut of the MIME-tree. A CP-leaf must be a consolidated blocking formula of the cut, because it contains only the literals in $pos(P) \cup any(Q)$. Since MIME-derivation does not select any literal in $neg(P) \cup any(Q)$, these literals appear in all the descendants goals, including the leaf nodes. Therefore, a consolidated blocking formula of any cut is also that of the cut which consists of all the leaf nodes. $\Pi^\circ \models_S \forall \boldsymbol{x}(W \rightarrow \neg p(\boldsymbol{t}))$ follows $\Pi^\circ \models \forall \boldsymbol{x}(W \rightarrow \neg p(\boldsymbol{t}))$. □

Example 3.18 The following is a CP-refutation of $\Pi_{cp} \cup \{\leftarrow \neg P(t)\}$, where Π_{cp} is the same as in Example 3.3:

G_0: $\leftarrow \neg p(t)$,
G_1: $\leftarrow [\neg p(t)], \neg q(f(t), t), \neg e(t, a), z(t)$
G_2: $\leftarrow \neg q(f(t), t), \neg e(t, b), \neg e(t, a)$,

where C_1 is used as input CP-chain from G_0 to G_1, while C_2 is used as input CP-chain from G_1 to G_2. It is easy to see that:

$$\Pi^\circ \models \neg q(f(t), t) \wedge \neg e(t, b) \wedge \neg e(t, a) \rightarrow \neg p(t).$$

□

In Section 3.5 we mentioned that for a blocking formula B of a cut Ξ in a MIME-tree of $SF(\Pi) \cup \{\leftarrow p(t)\}$, there does not necessarily exist a goal G_i in a CP-derivation of $\Pi_{cp} \cup \{\leftarrow \neg p(t)\}$ such that G_i is equal to B. To prove the completeness of CP-resolution, we rule out all the cuts in the MIME-tree which are irrelevant to the CP-derivation. With a cut Ξ in the MIME-tree and the blocking formula B of Ξ, the relevant cut Ξ' which is one level lower than Ξ should be unique, so that each blocking formula of any relevant cut Ξ in a MIME-tree of $SF(\Pi) \cup \{\leftarrow p(t)\}$ could be mapped to a goal G_i in some CP-derivation of $\Pi_{cp} \cup \{\leftarrow \neg p(t)\}$.

To achieve the goal, we need to reorder the MIME-tree, and manipulate the cuts as stated in the following algorithm. Note, the algorithm is only conceptually needed, to facilitate the completeness proof.

Algorithm 3.10

1. Reorder each direct children (and therefore the corresponding subtrees) for every node in the MIME-tree such that:
   (a) The children obtained through reduction inference rule are always to the right of the ones obtained through extension inference rule;
   (b) The child obtained through extension with a chain C' is to the right of another child obtained through extension with a chain C if C is ahead of C' (refer to Section 3.1 for the ordering of the chains).
2. If a node N in a cut Ξ is consolidatedly blocked by the concerned CP-goal, then move down to the cut Ξ' which is obtained from Ξ by replacing N with all the leaf nodes (or the node at the infinity if the branch is infinite) which are its descendants.
3. For each cut Ξ, the only relevant cut Ξ', which is one level lower than Ξ, is obtained from Ξ by moving down one level from the the rightmost nonleaf node N in Ξ.

□

Because a MIME-tree is not an ordered tree, at any node, the flipping of the order of its children, and thus the subtrees, would not affect the finiteness of the tree. As long as we have a single finite MIME-tree, we can always reorder the subtrees and not change the finiteness of the MIME-tree.

Theorem 3.11 (Completeness of CP-resolution) *Let Π be a set of clauses which possesses the weakly bounded term size property. Then for each chain*

$$C = \neg p(t) \leftarrow W$$

in $\Pi^{\circ}\downarrow p(t)$, there exist CP-leaves

$$G^1_{n_1} = \{\leftarrow W, E^1\}, \quad \ldots, \quad G^m_{n_m} = \{\leftarrow W, E^m\},$$

through CP-refutations of $\Pi_{cp} \cup \{\leftarrow \neg p(t)\}$, where $E^1, \ldots, E^m$ are equality formulas, such that

$$\mathrm{CET} \models \bigvee_{i=1}^{m} E^i.$$

□

Proof: Assume the finite MIME-tree is ordered in Algorithm 3.10. And cuts are manipulated as stated in Algorithm 3.10 too. We prove the theorem by proving that for each consolidated blocking formula $\{\neg d_1, \ldots, \neg d_q\}$ of the cut which consists of all the leaf nodes in the MIME-tree of $SF(\Pi) \cup \{\leftarrow p(t)\}$, there exists a CP-leaf $\{\leftarrow \neg d_1, \ldots, \neg d_q\}$ through a CP-refutation of $SF(\Pi) \cup \{\leftarrow p(t)\}$.

We can prove this by proving inductively on the level of the cut that each blocking formula of the cut of level k is reachable by a CP-derivation in k steps.

First, $k = 1$, i.e., the cut Ξ_1 contains only the root node, it is trivially true. Now assume the claim holds up to level k. We are going to prove the claim holds on $k+1$.

By induction hypothesis, for each blocking formula $\{\neg d_1, \ldots, \neg d_q\}$ of the cut Ξ_k, there is a CP-goal $G_{k-1} = \{\leftarrow \neg d_1, \ldots, \neg d_q\}$ through a CP-derivation $G_0 = \{\leftarrow p(t)\}, G_1, \ldots, G_{k-1}$ of $SF(\Pi) \cup \{\leftarrow p(t)\}$. Assume the cut Ξ_{k+1}, which is one level lower than Ξ_k, is obtained from Ξ_k by replacing the node $N \in \Xi_k$ with all its children nodes $\{N_1, \ldots, N_s\}$. Also assume the selected literal in N with respect to the MIME-derivation is ρ. Due to the Algorithm 3.10, the blocking formulas of the cut Ξ_{k-1} containing $\neg\rho$ must be in the form $\{\leftarrow \neg d_1, \ldots, \neg d_{i-1}, \neg\rho, \neg d_{i+1}, \ldots, \neg d_q\}$, with $\neg\rho = \neg d_i$, $1 \leq i \leq q$, as the rightmost literal in $neg(P) \cup any(Z)$. The blocking formulas of $\{N_1, \ldots, N_s\}$ must be in the form $\{\neg\rho_1, \ldots, \neg\rho_s\}$, such that ρ_j could be any newly introduced literal in N_j, for $1 \leq j \leq s$. This is because by definition, for every combinations of the literals $\{\rho_1, \ldots, \rho_s\}$ introduced into N_1 up to N_s through extension inference rule in MIME-derivations, there exists a CP-chain as the input chain of G_k which contains the literals $\{\neg\rho_1, \ldots, \neg\rho_s\}$; if ρ_j is introduced into N_j through tightening in MIME-derivation, $\neg\rho_j$ could be introduce into G_k through CP-tightening in CP-derivation; if ρ_j is introduced into N_j

through reduction inference rule in MIME-derivation, $\neg\rho_j$ must be added into G_k through CP-reduction in CP-derivation. Therefore all the possible combinations of the blocking formulas of $\{N_1, \ldots, N_s\}$ can be reached by G_k, which is in the form $G_n = \{\leftarrow \neg d_1, \ldots, \neg d_{i-1}, [\neg\rho], \neg\rho_1, \ldots, \neg\rho_s, \neg d_{i+1}, \ldots, d_q\}$. This also proves that all the possible combination of the blocking formula of level k cut can be reached by CP-derivation in k steps, for any finite k. Therefore each consolidated blocking formula of the cut which contains all the leaf nodes can be reached in finite steps through a CP-refutation of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$. □

In general, $m \neq 1$, because the equality formulas E^1 through E^m could be introduced from the equality literals of the failed goals.

We wish to emphasize here another property CP-resolution possesses, namely, it is strictly linear, and therefore, is one-staged.

The set of CP-constructed chains could be complete or not, depending on whether the given theory possesses the weakly bounded term size property, but one thing is for sure, every CP-constructed chain is sound.

Chapter 4

Extracting Answers in Circumscription

Circumscription is a nonmonotonic reasoning formalism proposed by McCarthy [31, 30] based on the notion of truth in all minimal models of a first-order theory. It is one of the most powerful among all those formalisms aimed at characterizing the nonmonotonic aspect of human thinking. Circumscription is known to be difficult in computation, because of the second-order quantifier involved in its syntactical definition. The existing algorithms [18, 36] are all multi-staged, which requires the given set of clauses to be decidable. Due to the fact that first-order logic is semi-decidable, those algorithms basically deal with only ground theories. As a direct consequence, they give only a YES or NO answers. In this chapter, we develop a proof procedure called CIR-resolution, which can extract answers in circumscription. CIR-resolution is ME-resolution augmented with the *completion* inference rule as nonmonotonic operator, which uses the proof-theoretic framework CP-resolution to construct the circumscription axiom into a set of clauses. If the given set of clauses possesses the weakly bounded term size property, then CP-resolution is complete, so is CIR-resolution. CP-resolution is a linear resolution, thus making CIR-resolution one-staged. This enables it to return a partial answer even the completion condition is not satisfied.

4.1 CIR-resolution: A Proof Procedure to Extract Answers in Circumscription

CIR-resolution is a proof procedure based on the proof-theoretic framework CP-resolution proposed in Chapter 3. It is used to extract answers in circumscription.

CIR-resolution is a ME-resolution augmented by CP-resolution. Due to the fact that ME-resolution is sound and complete, also because that CP-resolution is sound, and is complete when the given set of clauses possesses the weakly bounded term size property, CIR-resolution is sound, and is complete on the condition that the given set of clauses possesses the weakly bounded term size property.

Definition 4.1 Let Π be a set of clauses and $GF(\Pi)$ be its generalized full program, and Π_{cp} its blocking formulas. Let G be the *top goal.* Then *CIR-derivation* of a goal G_n from $GF(\Pi) \cup \Pi_{cp} \cup \{G\}$ is a sequence $G = G_0, G_1, \ldots, G_n$ of goals, such that

1. for $1 < i \leq n$, G_i is either
   (a) an extension goal of G_{i-1} and some chain C_i in $GF(\Pi)$, followed by a tightening operation, or
   (b) a deduction goal of G_{i-1}; or
   (c) a factorization of G_{i-1}; or
   (d) a completion goal of G_{i-1} and a CP-constructed chain C_i from Π_{cp} or its variant C_i';
2. perform a merging, followed by a truncation (ignore E); and
3. the equality formula of G_i is satisfiable;
4. G_i is acceptable.

A *CIR-refutation* of $GF(\Pi) \cup \Pi_{cp} \cup \{G\}$ is a CIR-derivation of a pseudo-empty clause of $GF(\Pi) \cup \Pi_{cp} \cup \{G\}$. □

4.2 Soundness and Completeness

Now we prove that CIR-resolution is sound with respect to the semantics of circumscription.

Theorem 4.1 (Soundness of CIR-resolution) *Let a CIR-refutation of* $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Phi\}$ *end up with a pseudo-empty goal* $\leftarrow E$. *Then the following holds:*

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} S_g(E) \rightarrow \Phi),$$

where $\boldsymbol{y}$ *is a tuple of variables in* E *disjoint to* $\boldsymbol{x}$, *and* $\boldsymbol{z}$ *is the tuple of ex-variables introduced in S-generalization.* □

Proof: By the fact ME-resolution is sound, together with Theorem 3.9 that $\Pi^{\circ} \models_S C$ for each CP-constructed clause C, we have:

$$\Pi \cup \Pi^{\circ} \models_S \forall(E \rightarrow \Phi).$$

By Theorem 3.8,

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}, \boldsymbol{y} \exists \boldsymbol{z}(S_g(E) \rightarrow \Phi),$$

which is

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} S_g(E) \rightarrow \Phi).$$

□

Example 4.1 Let the background theory Π contain the following clauses:

$C1$: $Makes(x, y) \rightarrow Robot(y)$;
$C2$: $Human(x) \vee Robot(x)$;
$C3$: $Human(f(x)) \rightarrow Human(x)$;
$C4$: $Makes(a, b)$,

such that $Makes, Robot \in pos(P)$ and $Human \in pos(Z)$. Q is empty to simplify the problem. The function f can be deemed as *father*. We now use P_1 to replace *Robot*, P_2 for *Makes*, and Z_1 for *Human*. And then rewrite Π as follows:

$C1$: $P_2(x, y) \rightarrow P_1(y)$;
$C2$: $Z_1(x) \vee P_1(x)$;
$C3$: $Z_1(f(x)) \rightarrow Z_1(x)$;
$C4$: $P_2(a, b)$,

The given top goal to the CIR-derivation is $\leftarrow Z_1(x)$.
$GF(\Pi)$ contains the following chains:

C_1^1 : $Z_1(x) \leftarrow \neg P_1(x)$;
C_2^1 : $Z_1(x) \leftarrow Z_1(f(x))$;
C_3^1 : $\neg Z_1(y) \leftarrow e(y, f(x)), \neg Z_1(x)$;
.....

$SF(\Pi)$ is omitted, and Π_{cp} contains the following chains:

C_1^2 : $\neg P_1(x) \leftarrow Z_1(x), \neg P_2(g_1(x_0), x)$;
C_2^2 : $\neg P_2(x, y) \leftarrow \neg e(\langle x, y \rangle, \langle a, b \rangle)$;
C_3^2 : $Z_1(x) \leftarrow \neg e(x, f(g_2(x_0)))$;
C_4^2 : $Z_1(x) \leftarrow Z_1(g_2(x_0))$;
.....

where g_1 and g_2 are Skolem functions introduced in $SF(\Pi)$.
CIR-derivation of $\Pi \cup \Pi_{cp} \cup \{\leftarrow Z_1(x)\}$ is the following sequence of goals:

Goals		input chain
G_0 :	$\leftarrow Z_1(x)$;	
G_1 :	$\leftarrow [Z_1(x)], \neg P_1(x)$;	C_1^1

At this moment, we need an CP-constructed chain. We thus carry on the following CP-derivation of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \neg P_1(x)\}$:

Goals		CP-chain
G_0' :	$\leftarrow \neg P_1(x_0)$	C_1^2
G_1' :	$\leftarrow [\neg P_1(x_0)], Z_1(x_0), \neg P_2(g_1(x_0), x_0)$	C_2^2
G_2' :	$\leftarrow [\neg P_1(x_0)], \neg e(\langle g_1(x_0), x_0\rangle, \langle a, b\rangle), Z_1(x_0)$	C_3^2
G_3' :	$\leftarrow \neg e(\langle g_1(x_0), x_0\rangle, \langle a, b\rangle), \neg e(x_0, f(g_2(x_0)))$.	

Therefore the CP-constructed chain is:

$$C_1^3 : \quad \neg P_1(x_0) \leftarrow \neg e(\langle g_1(x_0), x_0\rangle, \langle a, b\rangle), \neg e(x_0, f(g_2(x_0))),$$

which is to be used to carry on the CIR-derivation. Note the MIME-tree is infinite due to the recursive clause C_3 in Π.

The CIR-derivation is continued as follows:

Goals		input chain
G_2 :	$\leftarrow \neg e(\langle g_1(x), x\rangle, \langle a, b\rangle), \neg e(x, f(g_2(x)))$	C_1^3

G_2 is a pseudo-empty clause. By Theorem 4.1, we have:

$$CIR(\Pi; P; Z) \models \forall x(\forall z_{g_1(x)}, z_{g_2(x)}((\neg e(\langle z_{g_1(x)}, x\rangle, \langle a, b\rangle) \wedge \neg e(x, f(z_{g_2(x)}))) \rightarrow Z_1(x)). \tag{4.1}$$

and the computed answer is

$$\forall x \forall z_{g_1(x)}, z_{g_2(x)}((\neg e(\langle z_{g_1(x)}, x\rangle, \langle a, b\rangle) \wedge \neg e(x, f(z_{g_2(x)}))). \tag{4.2}$$

Formula (4.2) says, anyone (nobody is a or who is not b), and who is not a *father* of anyone else, is a human being.
The complete answer is apparently that *anyone who is not b, and is not a* forefather *of b, is a human being.* We fail to give a complete answer due to the infiniteness of the MIME-tree. □

Conceptually, CP-resolution is a subresolution of CIR-resolution. However, due the linearity of CP-resolution, it can be combined into CIR-resolution, and make a single linear resolution this way. This can seen from Example 4.1. We could insert $[Z_1(x)]$ into G'_0 through G'_3, and then alternatively number them as G_1 through G_4. Also name G_2 in the example G_5. Now we would have a single linear resolution.

To prove that CIR-resolution is complete under the weakly boundedness, we have to first prove it is true with respect to a ground query.

Lemma 4.2 (Ground Completeness of CIR-resolution) *Let Π be a set of clauses possessing the weakly bonded term property. Let $\leftarrow \Psi$ be a ground goal. If $\Pi \cup \Pi^\circ \models_S \Psi$, then there must exist the CIR-refutations of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\}$ which ends up with pseudo-empty clauses $\leftarrow E_1, \ldots, \leftarrow E_i, \ldots$, and there further exist substitutions*
$\theta_1, \ldots, \theta_i, \ldots$, such that

$$\Pi \cup \Pi^\circ \models_S \forall \boldsymbol{z} \bigvee_i S_g(E_i)\theta_i,$$

where each $S_g(E_i)$ is an S-generalization of E_i, and $\boldsymbol{z}$ is a tuple of variables introduced in S-generalization. □

Proof: See Appendix. □

Now we are ready to present the most important result in this chapter. Theorem 4.3 states if the given set of clauses Π possesses the weakly term bounded size property, then CIR-resolution can be used to extract answers in the form of an equality formula, and which is equivalent to the given query with respect to the semantics of circumscription and CET.

Theorem 4.3 (Completeness of CIR-resolution) *Let Π be a set of clauses possessing the weakly bounded term size property, and $G = \leftarrow \Psi$ be a goal. Suppose we have all the CIR-refutations of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\}$, which end up with pseudo-empty clauses $\leftarrow E_1, \ldots, \leftarrow E_i, \ldots$, then the following holds*

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i) \leftrightarrow \Psi),$$

where each $S_g(E_i)$ is an S-generalization of E_i with variables $\boldsymbol{z}$ introduced; $\boldsymbol{x}$ is a tuple of variables in Ψ, and $\boldsymbol{y}$ is a tuple of variables in $E_1, \ldots, E_i, \ldots$ disjoint to $\boldsymbol{x}$. □

Proof: $\Rightarrow$ This is exactly Theorem 4.1.
$\Leftarrow$ To prove

$$CIR(\Pi; P; Z) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i) \leftarrow \Psi)$$

we need to prove that

$$CIR(\Pi; P; Z) \models \exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i)\sigma \tag{4.3}$$

for each ground substitution σ such that $CIR(\Pi; P; Z) \models \Psi\sigma$.

By Theorem 3.7, $\Pi \cup \Pi^\circ \models_S \Psi\sigma$ if $CIR(\Pi; P; Z) \models \Psi\sigma$. By Lemma 4.2, there must exist CIR-refutations of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\sigma\}$, which end up with pseudo-empty clauses $\leftarrow E_1^*, \ldots, \leftarrow E_i^*, \ldots$, and there further exist substitutions $\theta_1, \ldots, \theta_i, \ldots$ such that

$$\Pi \cup \Pi^\circ \models_S \forall \boldsymbol{z} \bigvee_i S_g(E_i^*)\theta_i, \tag{4.4}$$

where each $S_g(E_i^*)$ is an S-generalization of E_i^* with variables $\boldsymbol{z}$ introduced.

By Lifting Lemma, there must exist CIR-refutations of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\}$, which end up with pseudo-empty clauses $\leftarrow E_1, \ldots, \leftarrow E_i, \ldots$, and there further exist non-Skolem substitutions $\lambda_1, \ldots, \lambda_i, \ldots$ and $\sigma_1, \ldots, \sigma_i, \ldots$, such that

$$E_i\lambda_i = E_i^*, \tag{4.5}$$

and

$$\lambda_i = \sigma\sigma_i. \tag{4.6}$$

From (4.4), (4.5), (4.6) and the fact that $S_g(E_i\lambda_i) = S_g(E_i)\lambda_i$ for non-Skolem substitution λ_i, we have

$$\Pi \cup \Pi^\circ \models_S \forall \boldsymbol{z} \bigvee_i S_g(E_i)\sigma\sigma_i\theta_i. \tag{4.7}$$

By Theorem 3.7, we have

$$CIR(\Pi; P; Z) \models \exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i)\sigma. \tag{4.8}$$

Since σ replaces $\boldsymbol{x}$ with ground terms, the domain of $\sigma_i\theta$ must be the variables disjoint to $\boldsymbol{x}$, therefore (4.3) is proved, and thus, the theorem. □

CIR-resolution is one-staged. By the definition, CIR-resolution is basically ME-resolution augmented with completion inference rule. Completion inference rule is actually an extension using a CP-constructed chain or its variant through CP-resolution. Both ME-resolution and CP-resolution are one-staged, which make CIR-resolution to be one-staged too. Due to its one-stagedness, CIR-resolution can extract partial answers even when the completion condition is not met.

4.3 Extended Results

The results presented up to this point are based on the assumption that the declarative semantics is *parallel circumscription*. Another assumption is that the given query is a conjunction of literals free of quantifier. We made these assumptions only for the sake of clarity. They are by no means indispensable.

4.3.1 CIR-resolution for Prioritized Circumscription

The minimization of the predicates in P conflict one another in some of the applications, and John McCarthy [30] introduced a priority, so that certain predicates have a higher privilege to be minimized than the others in case a conflict occurs. *Prioritized circumscription* is the expression:

$$CIR\,(\Pi; P^1 > \cdots > P^k; Z),$$

where $P^1, \ldots, P^k$ is a partition of P with P^i having a higher priority to be minimized than P^j, for $1 \leq i < j \leq k$.

Lifschitz [26] rewrote the prioritized circumscription in an equivalent expression, which is a conjunction of parallel circumscriptions:

$$CIR\,(\Pi; P^1 > \cdots > P_k; Z) \equiv \bigwedge_{i=1}^{k} CIR\,(\Pi; P^i; P^{i+1}, \ldots, P^k, Z).$$

Intuitively, in each parallel circumscription $CIR\,(\Pi; P^i; P^{i+1}, \ldots, P^k, Z)$, P^i is the set of predicates to be minimized, $P^{i+1} \cup \cdots \cup P^k$ join Z and become the sets of predicates to vary, while $P^1 \cup \cdots \cup P^{i-1}$ join Q and become the fixed predicates.

In terms of MIME-resolution, the literals in fixed predicates and the negation of the predicates to be minimized are the ones to be skipped.

For $1 \leq i \leq k$, let Π_i^* be obtained in the same way as Π, except that fixing $Q \cup P^1 \cup \cdots \cup P^{i-1}$, minimizing only P^i, and allowing $P^{i+1} \cup \cdots \cup P^k \cup Z$ to vary. By Theorem 3.4,

$$CIR\,(\Pi; P^i; P^{i+1}, \ldots, P^k, Z) \equiv \Pi \cup \Pi_i^*.$$

Therefore the following equation holds:

$$CIR\,(\Pi; P^1 > \cdots > P_k; Z) \equiv \Pi \cup \bigcup_{i=1}^{k} \Pi_i^*.$$

Next, use CP-derivation to construct as many chains in Π_i° as possible. If the background theory possesses the weakly bounded term size property with respect

to that *ith-level* MIME-resolution, then every chain in Π_i° can be CP-constructed. With each Π_i° constructed, the queries can be answered the same way as stated in Section 4.2.

The relation priority can be a partial order. For instance we have the following prioritized circumscription:

$$CIR\,(\Pi; P^{1,1} > \cdots > P^{1,k_1}, \ldots, P^{n,1} > \cdots > P^{n,k_n}; Z),$$

among which P^{i_1,j_1} and P^{i_2,j_2} are incomparable if $i_1 \neq i_2$.

$\Pi^*_{i,j}$ can be obtained in the same way as Π^* except that fixing $Q \cup P^{i,1} \cup \cdots \cup P^{i,j-1}$, allowing $P^{i,j+1} \cup \cdots \cup P^{i,k_i} \cup Z$ to vary, while minimizing all the other predicate, and

$$\Pi^*_{i,j} = \{\Pi^*_{i,j} \downarrow p(\boldsymbol{t}) \mid p(\boldsymbol{t}) \in pos(P^{i,j})\}.$$

The above extended results are presented without proof. Actually they can be proved by means of the model theoretical definition of prioritized circumscription detailed in [26].

4.3.2 Answer Generic Query with CIR-resolution

Since our purpose is to extract answers in circumscription, we would like the query to be a quantifier free formula, but it does not have to be a conjunction of literals. Since every quantifier free formula has an equivalent disjunctive normal form [25], we assume it is given in disjunctive normal form, without loss of generality.

Suppose the given set of clauses is Π, and the query is $\bigvee_{i=1}^{n} W_i(\boldsymbol{x}_i)$, where $W_i(\boldsymbol{x}_i)$ is a conjunction of literals with variables $\boldsymbol{x}_i$. We introduce a new predicate $z_{qr}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in pos(Z)$. For simplicity we assume that z_{qr} is in the underlying language, but not in Π. This assumption is reasonable since we can always introduce a new predicate and expand the language without changing the nature of the issue. We define Π^+ as:

$$\Pi^+ = \Pi \cup \bigcup_{i=1}^{n} z_{qr}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \leftarrow W_i(\boldsymbol{x}_i),$$

and we have the following Lemma 4.4.

Lemma 4.4

1. *An interpretation I_1 is a model of $CIR(\Pi; P; Z)$ only if $CIR(\Pi^+; P; Z)$ has a model I_2 such that I_1 and I_2 differ only on how to interpret the ground atoms in z_{qr}.*

2. *An interpretation I is a model of $CIR(\Pi^+; P; Z)$ only if it is a model of $CIR(\Pi; P; Z)$.*

□

Proof: (1) Let I_2 be constructed as this, the assignment of each ground atom $\rho(\boldsymbol{a})$ in I_2 is the same as in I_1, except the assignment of each ground atom $z_{qr}(\boldsymbol{b})$, which is always true in I_2. By definition, we have $I_1 \leq I_2$ and $I_2 \leq I_1$. We need to prove that I_2 is a model of $CIR(\Pi^+; P; Z)$. Since I_1 is a model of Π, z_{qr} is not in Π, and I_1 and I_2 differ on only the assignment of z_{qr}, I_2 must be a model of Π too. Furthermore, I_2 is a model of $z_{qr}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \leftarrow W_i(\boldsymbol{x}_i)$, for $1 \leq i \leq n$, since each $z_{qr}(\boldsymbol{b})$ is true in I_2. I_2 is therefore a model of Π^+. Now prove that I_2 is a minimal model of Π^+. Assume $I_3 \models CIR(\Pi^+; P; Z)$, and $I_3 < I_2$. Which implies that $I_3 \models \Pi$, and $I_3 < I_1$. I_1 is thus not a model of $CIR(\Pi; P; Z)$. Therefore I_2 must be a model of $CIR(\Pi^+; P; Z)$.

(2) I is a model of $CIR(\Pi^+; P; Z)$ implies that it is a model of Π^+, which further implies that it is a model of Π, since Π is a subset of Π^+. We need only to prove that I is a minimal model of Π. Assume $I' \models CIR(\Pi; P; Z)$ and $I' < I$. By Lemma 4.4 (1), there must exist an interpretation I_2 such that $I_2 \models \Pi^+$ and $I_2 \leq I'$. It follows that $I_2 < I$, and I is not model a model of $CIR(\Pi^+; P; Z)$, contradicting the assumption. Therefore, I is a model of $CIR(\Pi; P; Z)$. □

Theorem 4.5 reformulates the given set of clauses and the query so that the query is reduced to a single literal.

Theorem 4.5 *$CIR(\Pi; P; Z) \models \bigvee_{i=1}^{n} W_i(\boldsymbol{x}_i)\theta$ if and only if $CIR(\Pi^+; P; Z) \models z_{qr}(\boldsymbol{x}_1, \boldsymbol{x}_n)\theta$ for each ground substitution θ such that $\boldsymbol{x}_i\theta$ is ground for $1 \leq i \leq n$.* □

Proof: $\Rightarrow$ Assume a model I of $CIR(\Pi^+; P; Z)$ is not a model of $z_{qr}(\boldsymbol{x}_1, \boldsymbol{x}_n)\theta$. We must have

$$I \models \neg W_i(\boldsymbol{x}_i)\theta$$

for $1 \leq i \leq n$, which implies

$$I \models \neg \bigvee_{i=1}^{n} W_i(\boldsymbol{x}_i)\theta.$$

By Lemma 4.4, I is a model $CIR(\Pi; P; Z)$. It follows that $CIR(\Pi; P; Z) \not\models \bigvee_{i=1}^{n} W_i(\boldsymbol{x}_i)\theta$.

$\Leftarrow$ Assume a model I_1 of $CIR(\Pi; P; Z)$ is not a model of $\bigvee_{i=1}^{n} W_i(\boldsymbol{x}_i)\theta$. By Lemma 4.4, $CIR(\Pi^+; P; Z)$ has a model I_2 such that I_1 and I_2 differ only on how to assign ground atoms in z_{qr}. Therefore

$$I_2 \models \neg W_i(\boldsymbol{x}_i)\theta$$

for $1 \leq i \leq n$. There must exist a model I_3 of $CIR(\Pi^+; P; Z)$ which has the same assignment for every ground atom as in I_2 except

$$I_3 \models \neg z_{qr}(\boldsymbol{x}_1, \boldsymbol{x}_n)\theta.$$

It follows that $CIR(\Pi^+; P; Z) \not\models z_{qr}(\boldsymbol{x}_1, \boldsymbol{x}_n)\theta$. $\square$

Theorem 4.5 indicates that CIR-resolution can be used to extract answer even the given query is in general form.

4.4 Comparison with Other Works

Bossu *et al* [3], Przymusinski [36], and Ginsberg [18] proposed their algorithms in a bid to answer the queries with respect to the semantics of parallel circumscription. Bossu's algorithm does not deal with the predicate set Z, and requires more restrictions on the given set of clauses [3, 36]. In an article aimed at analysis and comparison of the latter two algorithms [20], Inoue *et al* pointed out that both Przymusinski and Ginsberg established their results on the same theoretical basis. However, Ginsberg's theorem prover is more restrictive since it does not deal with the set of predicates in Q. We would like to focus on Przymusinski's *query evaluation algorithm* because it is a full fledged query answering mechanism for parallel circumscription.

Przymusinski's algorithm basically applies only to ground cases (so does Ginsberg's theorem prover). As a result, it gives only a YES or NO answer to the queries, rather than extracting answer such as 'What', 'When', 'Where' and 'How'. The reason he restricted it to ground cases is that the algorithm doesn't work with infinite universes.

CIR-resolution can do everything Przymusinski's algorithm can, but not the vice versa. We investigate all the reasons behind that. In this section, we describe how his algorithm works, and then explore why the algorithm has such limitation. We further describe what techniques we adopted to make all the differences.

4.4.1 Przymusinski's Query Evaluation Algorithm and MILO-Resolution

In [16], Gelfond *et al* proved that with a finite universe, parallel circumscription is equivalent to ECWA (extended closed world assumption).

Theorem 4.6 (Theorem 5.1 in [16]) *For any sentence Ψ we have:*

$$CIR(\Pi; P; Z) \models \Psi \text{ if and only if } \mathrm{ECWA}(\Pi; P; Z) \vdash \Psi.$$

□

The notion of ECWA is defined upon that of *free from negation*:

Definition 4.2 [Definition 4.3 in [16]] The *Extended Closed World Assumption* with respect to P and Z applied on the theory Π results in its closure $\mathrm{ECWA}(\Pi; P; Z)$ as follows:

$$\mathrm{ECWA}(\Pi; P; Z) = \Pi \cup \{\neg K : K \textit{ is free from negation in } \Pi\}.$$

□

Free from negation is defined as:

Definition 4.3 [Definition 4.1 in [16]] Let K be an arbitrary sentence not in $any(Z)$. K is *free from negation* in Π if there exists no disjunction of literals $B = \bigvee_i^n B_i$, where $B_i \in pos(P) \cup any(Q)$, such that:

1. $\Pi \models K \vee B$;
2. $\Pi \not\models B$.

□

The theoretical basis for Przymusinski's result is the following two theorems from [16] and [36] which are the applications of Theorem 4.6.

Theorem 4.7 (Theorem 2.5 in [36]) *Suppose that Ψ is a variable-free formula in $any(Q) \cup any(P)$. Then $CIR(\Pi; P; Z) \models \Psi$ if and only if there is no ground clause D such that*

1. *$D \in any(Q) \cup pos(P)$;*
2. *$\Pi \models \neg\Psi \vee D$, but $\Pi \not\models D$.*

□

Theorem 4.8 (Theorem 2.6 in [36]) *Suppose that Ψ is any ground formula. Then*

$$CIR(\Pi; P; Z) \models \Psi$$

if and only if either $\Pi \models \Psi$ or there is a ground formula G such that

1. $G \in any(Q) \cup pos(P)$;
2. $\Pi \models \Psi \vee G$;
3. $CIR(\Pi; P; Z) \models \neg G$.

□

The central part of Przymusinski's algorithm is MILO-*resolution*, which is obtained from OL-resolution [7] by skipping the literals from $pos(P) \cup any(Q)$ (Notice, should MILO also use oriented chains, it would have skipped the literals from $neg(P) \cup any(Q)$ just as MIME-resolution does). A MILO-*leaf* of $\Pi \cup \{C\}$ is a clause deducible using MILO-deduction with given set of clauses Π and top clause C, and containing no literals from $neg(P) \cup any(Z)$. The conjunction of all MILO-leaves of $\Pi \cup \{C\}$ is called MILO-*derivative* of $\Pi \cup \{C\}$, and is denoted as $Deriv(\Pi, C)$. For a set of clauses $H = \{C_1, \ldots, C_n\}$, $Deriv(\Pi, H)$ is defined as:

$$Deriv(\Pi, H) = \bigwedge_{i=1}^{n} Deriv(\Pi_i, C_i),$$

where $\Pi_1 = \Pi$, and $\Pi_{i+1} = \Pi_i \cup \{C_i\}$, for $i = 1, \ldots, n-1$.

We will not go into the details of the definition of MILO-resolution. In fact, MILO-resolution is similar to ground MIME-resolution with a few exceptions as follows:

1. MILO is based on ground OL-resolution [7], and MIME is based on weak version of ground ME-resolution;
2. MILO is in non-oriented form and uses input clauses, while MIME is in oriented form and uses general form input chains;
3. The top goal of MIME contains only a single literal in $pos(P)$, while MILO has no such restriction.

More precisely, MILO-resolution (as is MIME-resolution) is a deductive engine to generate all the MILO-derivatives (MIME-preconditions).

The following is Przymusinski's algorithm:

Algorithm 4.9 [Algorithm 4.1 in [36]] The query answering algorithm to decide whether a given formula Ψ is implied by circumscriptive theory $CIR(\Pi; P; Z)$:

Step 1 Check if the formula Ψ contains literals from Z or not. If it does not then go to Step 2, else go to Step 3.

Step 2 (there are no literals from Z)

1. Represent Ψ in conjunctive normal form, i.e., $\Psi = \bigwedge_{i=1}^{m} G_i$.
2. For all $i = 1, \ldots, m$, use Algorithm 4.10 to find $Deriv(\Pi, G_i)$.
3. For all $i = 1, \ldots, m$, use known resolution to verify whether $\Pi \models Deriv(\Pi, G_i)$.
4. If there is a i so that $\Pi \not\models Deriv(\Pi, G_i)$, then return answer NO, else return answer YES.

Step 3 (there are literals from Z).

1. Represent Ψ in disjunctive normal form, i.e., $\Psi = \bigvee_{i=1}^{m} K_i$.
2. Set $\Pi_1 = \Pi$ and for all $i = 1, \ldots, m$, use Algorithm 4.10 to find $Deriv(\Pi_i, \neg K_i)$, where $\Pi_{i+1} = \Pi_i \cup \neg K_i$.
3. Use Step 2 to verify whether

$$CIR(\Pi; P; Z) \models \neg \bigwedge_{i=1}^{m} Deriv(\Pi_i, \neg K_i).$$

4. If yes, then return the answer YES, else return the answer NO.

□

Algorithm 4.10 [Algorithm 4.2 in [36]] The algorithm to compute $Deriv(\Pi, C)$ for a given theory Π and a given clause C:

Step 1 Use the depth-first search on the MILO-resolution tree with the top clause C to find all the MILO-leaves of $\Pi \cup C$.

Step 2 Let $Deriv(\Pi, C)$ be the conjunction of all such MILO-leaves.

□

4.4.2 Weakness with Przymusinski's Query Evaluation Algorithm

Now assume the given query is $Qr = L_1 \wedge L_2 \wedge \cdots \wedge L_n$, or equivalently, the top goal is $\leftarrow L_1, L_2, \ldots, L_n$. Assume the query does not contain any literal in $any(Z)$. To find out whether $CIR(\Pi; P; Z) \models Qr$, we have to go through the following steps:

1. Since Qr is already in conjunctive normal form. By Step 2 (2) of Algorithm 4.9, for $i = 1, \ldots, n$, use MILE-resolution to find $Deriv(\Pi, L_i)$.

2. By Step 2 (3) of Algorithm 4.9, use known resolution to verify whether $\Pi \models Deriv(\Pi, L_i)$, or more accurately, use known resolution to find whether $\Pi \cup \neg Deriv(\Pi, L_i)$ is consistent. Note, here we assume that the MILO-tree is finite, $Deriv(\Pi, L_i)$ is computed, and negated, and then rearranged in conjunctive normal form, since $Deriv(\Pi, G_i)$ is in conjunctive normal form, and consequently $\neg Deriv(\Pi, G_i)$ is in disjunctive normal form.

3. If there is an i, $\Pi \not\models Deriv(\Pi, L_i)$, then return answer NO, else return answer YES.

Now let us assume the given query does contain some literals in $any(Z)$. To find out whether $CIR(\Pi; P; Z) \models Qr$, we go through the following steps:

1. Since Qr is already in disjunctive normal form, according to Algorithm 4.9 Step 3 (2), we have to first find $Deriv(\Pi, \neg Qr)$ using MILO-resolution.

2. Use Step 2 of Algorithm 4.9 to verify whether $CIR(\Pi; P; Z) \models \neg Deriv(\Pi, \neg Qr)$. But according to Step 2 (1) of Algorithm 4.9, we have to first represent $\neg Deriv(\Pi, \neg Qr)$ in conjunctive normal form, i.e., $\neg Deriv(\Pi, \neg Qr) = \bigwedge_i G_i$. Note, here we already assume that the MILO-tree is finite, $Deriv(\Pi, \neg Qr)$ is computed, and then negated, and then be rearranged in conjunctive normal form, since $Deriv(\Pi, G_i)$ is in conjunctive normal form, and consequently $\neg Deriv(\Pi, G_i)$ is in disjunctive normal form.

3. By Step 2 (2) of Algorithm 4.9, for all i, use MILO-resolution to find $Deriv(\Pi, G_i)$.

4. By Step 2 (3) of Algorithm 4.9, use known resolution to verify whether $\Pi \models Deriv(\Pi, G_i)$, or more accurately, use known resolution to find whether $\Pi \cup \neg Deriv(\Pi, G_i)$ is consistent. Note, here once again we assume that the MILO-tree is finite, $Deriv(\Pi, G_i)$ is computed, and negated, and then rearranged in conjunctive normal form.

5. If there is an i, $\Pi \not\models Deriv(\Pi, G_i)$, then return answer NO, else return answer YES.

We see that before we can finally have a YES or NO answer, we have to wait for the MILO-derivatives to be complete. For the query with literals in $any(Z)$, we have to wait twice. If in either case the MILO-tree is infinite, what we wait for will never happen, and there will never be an answer. Therefore, Przymusinski's query evaluation algorithm is a *3-staged* algorithm. It basically deals with finite ground cases (as opposed to the ground instantiation of a general theory to an infinite univers). Because the semidecidability of first-order logic, one should not expect that the resolution tree to be finite. MILO's answer returning policy is *all or nothing*. When all the MILO-trees are finite, it can return all the answers (note the MILO-derivatives have to be Skolemized before being negated); otherwise, none.

The following example is from [36].

Example 4.2 Let Π contains the following clauses:

C1: $\neg q_1 \vee p \vee z$;
C2: $\neg q_2 \vee \neg z$; and
C3: $\neg q_3 \vee \neg z$.

And we want to prove $CIR(\Pi; P; Z) \models G$ where $G = \neg q_1 \vee z \vee q_2 \vee q_3$. Since G contains literals from $any(Z)$, we use Step 3 in Algorithm 4.9, and find

$$Deriv(\Pi, \neg G) = q_1 \wedge (\neg q_1 \vee p) \wedge \neg q_2 \wedge \neg q_3.$$

Since the first conjunct is trivially true,

$$\neg Deriv(\Pi, \neg G) = \neg q_1 \vee \neg p \vee q_2 \vee q_3.$$

We use Step 2 of Algorithm 4.9, and find that $Deriv(\Pi, \neg Deriv(\Pi, \neg G))$ is empty, therefore

$$\Pi \models Deriv(\Pi, \neg Deriv(\Pi, \neg G))$$

and

$$CIR(\Pi; P; Z) \models G$$

□

Now let us see how CIR-resolution solves this problem.

Example 4.3 Assume Π contains the same set of clauses as Example 4.2. Since the query is a disjunction of literals, according to Subsection 4.3.2, we add a new predicate Z_{qr}, and the corresponding clauses:

C'4: $Z_{qr} \vee q_1$;
C'5: $Z_{qr} \vee \neg z$;
C'6: $Z_{qr} \vee \neg q_2$;
C'7: $Z_{qr} \vee \neg q_3$.

Also some simple CP-refutations would give the chains in Π°, they are:

C"8: $\neg p \leftarrow \neg q_1$;
C"9: $\neg p \leftarrow \neg q_2, \neg q_3$.

Now, here starts the CIR-derivation:

G_0:	$\leftarrow Z_{qr}$,	extension with C'4
G_1:	$\leftarrow [Z_{qr}], \neg q_1$,	extension with C1
G_2:	$\leftarrow [Z_{qr}], [\neg q_1], \neg p, \neg z$,	extension with C'5
G_3:	$\leftarrow [Z_{qr}], [\neg q_1], \neg p, [\neg z], \neg Z_{qr}$,	reduction
G_4:	$\leftarrow [Z_{qr}], [\neg q_1], \neg p$,	completion with C"9
G_5:	$\leftarrow [Z_{qr}], [\neg q_1], [\neg p], \neg q_2, \neg q_3$,	extension with C'7
G_6:	$\leftarrow [Z_{qr}], [\neg q_1], [\neg p], \neg q_2, [\neg q_3], \neg Z_{qr}$,	reduction
G_7:	$\leftarrow [Z_{qr}], [\neg q_1], [\neg p], \neg q_2$,	extension with C'6
G_8:	$\leftarrow [Z_{qr}], [\neg q_1], [\neg p], [\neg q_2], \neg Z_{qr}$,	reduction
G_9:	$\leftarrow \square$.	

We have proved $CIR(\Pi; P; Z) \models G$ using CIR-resolution. Note that CIR-resolution is designed to take the query in the form of a conjunction of literals, it is to our disadvantage to compute a query in a disjunction of literals as in this example. $\square$

Now we give some examples where Przymusinski's algorithm doesn't work. To simplify the issue, we give the query which is not in $any(Z)$. The convenient format of the query to Przymusinski's algorithm is a disjunction of literals. However, since the query format of a conjunction of literals is well accepted in logic programming, we do not see any point to compromise. Suppose the query is $G = L_1, \ldots, L_n$. Przymusinski's algorithm has to compute $Deriv(\Pi, L_i)$ for $1 \leq i \leq n$. However, for the literal $L_i \in pos(P) \cup any(Q)$, $Deriv(\Pi, L_i) = L_i$. Therefore, it only computes the $Deriv(\Pi, L_i)$, where $L_i \in neg(P)$, the rest is just the verification of consistency. This somewhat coincides with CP-resolution. Therefore, we decide to give a single literal query which is in $neg(P)$.

Example 4.4 This example is the same as Example 3.15 and 3.12. Let Π contains the following clauses:

C1: $p(x,y) \vee \neg z(x) \vee \neg p(f(x),a)$,
C2: $z(b)$.

Let the query be $\neg p(x,y)$. Although if we flip $z(x)$ and $\neg p(f(x),a)$ in C1, Algorithm 4.9 would turn out the correct answer, however, as shown in the following deduction, it has no ability to pick up the correct order. By Step 2 (2) of Algorithm 4.9, use MILO-resolution to find out $Deriv(\Pi, \neg p(x,y))$.

G_0: $\neg p(x,y)$;
G_1: $\neg z(x) \vee \neg p(f(x),a)$;
G_2: $\neg z(x) \vee \neg z(f(x)) \vee \neg p(ff(x),a)$;
.....

As we see in Example 3.12, CP-resolution and therefore CIR-resolution can turn out a sound and complete answer which is TRUE, while Przymusinski's algorithm fails to terminate. □

The above example shows that Przymusinski's algorithm requires that the given theory possesses the strong bounded term size property.

The following example shows that it won't turn out partial answer as CP-resolution and CIR-resolution do.

Example 4.5 This is the same as Example 3.13. Let Π contain $\{p(x) \vee \neg q(x) \vee \neg p(f(x)); \ \neg q(a)\}$. Let the query be $\neg p(x)$. By Algorithm 4.9 Step 2 (2), use MILO-resolution to find out $Deriv(\Pi, \neg p(x))$.

G_0: $\neg p(x)$;
G_1: $\neg q(x) \vee \neg p(f(x))$;
G_2: $\neg q(x) \vee \neg q(f(x)) \vee \neg p(ff(x))$;
.....

The MILO-derivation will never terminate. However in Example 3.13, CP-resolution, and therefore CIR-resolution can turn out a partial answer which is $e(x,a)$. □

As we pointed out earlier, the convenient format of the query to Przymusinski's algorithm is $L_1 \vee \cdots \vee L_i \vee \cdots \vee L_n$. However, the algorithm doesn't have the intelligence to know that $CIR(\Pi; P; Z) \models L_1 \vee \cdots \vee L_i$ implies that $CIR(\Pi; P; Z) \models L_1 \vee \cdots \vee L_i \vee \cdots \vee L_n$. Therefore we can always do some seemingly harmless thing and make the algorithm doesn't work as shown in the following example. While CIR-resolution certainly does not have this weakness.

Example 4.6 Suppose $CIR(\Pi; P; Z) \models \Psi$. We add to Π an irrelevant clause so that $\Pi^+ = \Pi \cup \{p(x) \leftarrow p(f(x))\}$, and let the query be $\Psi^+ = \Psi \vee \neg p(x)$, where $p \in pos(P)$ is a predicate symbol not in Π. Because it has to wait for an infinite MILO-branch to finish, the algorithm can never confirm $CIR\ (\Pi^+; P; Z) \models \Psi^+$, although Π^+ is stronger than Π while Ψ^+ is weaker than Ψ, □

We are not going to argue that $\{\neg K : K$ *is free from negation in* $\Pi\}$ in Definition 4.2 is a larger set than Π^*, although it is true, because that is not the purpose of this dissertation.

The reason which causes all the problems with Przymusinski's algorithm is the multi-stageness. On the other hand, it is CP-resolution on our part which makes all the difference. We merely view MIME-resolution, which is the counterpart of MILO-resolution, as a bridge leading to CP-resolution. We do not wait until MIME-resolution has produced every MIME-leaf, but use CP-resolution to construct the chains while the imaginary MIME-resolutions are still going on. CP-resolution has a built-in predicate completion functionality, and is thus able to construct a subset of $\neg p \leftarrow \neg Prec(\Pi, \{\leftarrow p\})$. Although the constructed chains may sometimes be weaker than what are exactly in $\neg p \leftarrow \neg Prec(\Pi, \{\leftarrow p\})$. But if the weaker ones can do the job, why insist the stronger ones? In case the given set of clauses possesses the weakly bounded term size property, when Przymusinski's algorithm still might not work, CIR-resolution is guaranteed to be a sound and complete proof procedure, or a sound and complete query answering system.

Chapter 5

Computing Logic Programs

In this chapter, we investigate the existing semantics for logic programming, the *well-founded model* [12] and the *stable model* semantics [14], and how they can be redefined in terms of circumscription with autoepistemic theories [48]. We then develop proof procedures for these two semantics based on the CP-resolution proof-theoretic framework.

5.1 Two Semantics: Stable and Well-founded

The well-founded model semantics, proposed by Van Gelder, Ross and Schlipf in 1988 [12], and the stable model semantics, proposed by Gelfond and Lifschitz in 1988 [14], are two of the most prominent semantics ever proposed for logic programs with negation.

Both stable and well-founded semantics can be characterized by the fixpoint and the alternating fixpoint of the Gelfond-Lifschitz (GL) transformation introduced in [14]. In fact, an interpretation I is a stable model of the program $\mathcal{P}$ if and only if it is a fixpoint of the GL-transformation of $\mathcal{P}$, while the well-founded semantics is determined by the set of all alternating fixpoints, i.e., fixpoints of the double GL-transformation, of the program $\mathcal{P}$.

A logic program is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$$

where $m, n \geq 0$ and a, b_i's, and c_j's are atoms.

Let $\mathcal{P}$ be a program and I be a (two-valued) Herbrand interpretation of $\mathcal{P}$. Then the Gelfond-Lifschitz transformation of $\mathcal{P}$ with respect to I is the logic program $\mathcal{P}^I$ obtained from $\mathcal{P}$ as follows:

1. eliminate from $\mathcal{P}$ each rule whose body contains the negation of an atom in I;

2. from the body of each remaining rule in $\mathcal{P}$, delete all negative literals.

Recall the transformation $\mathbf{T}_P(I)$, called the *immediate consequence operator for a Horn program*, whose output is a set of atoms such that $a \in \mathbf{T}_P(I)$ if and only if a is the head of some rule in $\mathcal{P}$ all of whose literals in the body are in I. $\mathcal{P}^I$ is a Horn program and hence has a unique least model which is given by $\mathbf{T}_{\mathcal{P}^I} \uparrow \omega$.

As in [48], we define $\mathbf{S}_\mathcal{P}(I) = \mathbf{T}_{\mathcal{P}^I} \uparrow \omega$. A *fixpoint* of $\mathbf{S}_\mathcal{P}$ is an interpretation I of $\mathcal{P}$ such that $I = \mathbf{S}_\mathcal{P}(I)$. A fixpoint of $\mathbf{S}_\mathcal{P}$ is also called a fixpoint of $\mathcal{P}$.

Definition 5.1 ([14, 46]) I is a stable model of $\mathcal{P}$ if and only if I is a fixpoint of $\mathbf{S}_\mathcal{P}$. □

One of the problems here is that $\mathbf{S}_\mathcal{P}$ may not always have fixpoints. Van Gelder proposed the alternating fixpoint to resolve the problem. The basic idea is that $\mathbf{S}_\mathcal{P}$ may not always have fixpoints, but $\mathbf{S}^2_\mathcal{P}(I)$, the double GL-transformation, does.

An *alternating fixpoint* of program $\mathcal{P}$ is an interpretation I of $\mathcal{P}$ such that

$$I = \mathbf{S}^2_\mathcal{P}(I).$$

Let $\mathbf{A}_\mathcal{P}(I)$ be $\mathbf{S}^2_\mathcal{P}(I)$. By [46, 1], $\mathbf{S}_\mathcal{P}$ is antimonotonic, that is, $\mathbf{S}_\mathcal{P}(I) \subseteq \mathbf{S}_\mathcal{P}(J)$ if $J \subseteq I$, and $\mathbf{A}_\mathcal{P}$ is monotonic, whose least fixpoint is given by $\mathbf{A}_\mathcal{P} \uparrow \omega$. Therefore, every logic program has at least one alternating fixpoint.

Proposition 5.1 ([46]) *Let T be $\mathbf{A}_\mathcal{P} \uparrow \omega$, the least fixpoint of $\mathbf{A}_\mathcal{P}$, $F = \{a | a \notin \mathbf{S}_\mathcal{P}(T)\}$. Then $< T, F >$ is the well-founded model of $\mathcal{P}$.* □

Theorem 5.2 and Corollary 5.3 are from [12] which give the relationship between stable model and well-founded partial model, and stable model and well-founded model, respectively.

Theorem 5.2 (Theorem 6.1 in [12]) *Any stable model M is an extension of the well-founded partial model $I^\star$.* □

And therefore the following corollary holds.

Corollary 5.3 (Corollary 6.2 in [12]) *If $\mathcal{P}$ has a two-valued well-founded model, then that model is the unique stable model.* □

We use $ST(\mathcal{P})$ and $WF(\mathcal{P})$ to denote the stable and well-founded semantics of the program $\mathcal{P}$. By Theorem 5.2, the following holds.

$$WF(\mathcal{P}) \models \Psi \quad \textit{only if} \quad ST(\mathcal{P}) \models \Psi,$$

but not vice versa.

Gelfond has first proposed that a logic program with negation be represented as an autoepistemic theory by replacing negative literals $\neg c$ with $\neg \mathcal{L}c$, standing for *believing in* $\neg c$ [13].

Definition 5.2 Assume $\mathcal{P}$ is a logic program with a set P of predicates. Let $\mathcal{L}P$ be the set of belief predicates whose negation standing for "*believing in* $\neg P$". Then $\mathcal{P}(P, \mathcal{L}P)$ is used to represent the corresponding autoepistemic theory (also called the belief theory) consisting of rules

$$a \leftarrow b_1 \wedge \cdots \wedge b_n \wedge \neg \mathcal{L}c_1 \wedge \cdots \wedge \neg \mathcal{L}c_m.$$

□

Example 5.1 $\mathcal{P} = \{a \leftarrow \neg b;\ \ p \leftarrow a, \neg q;\ \ q \leftarrow b, \neg p\}$ can be represented by a belief theory $\mathcal{P}(P, \mathcal{L}P) = \{a \leftarrow \neg \mathcal{L}b;\ \ p \leftarrow a, \neg \mathcal{L}q;\ \ q \leftarrow b, \neg \mathcal{L}p\}$. Note that $P = \{a, b, p, q\}$ and $\mathcal{L}P = \{\mathcal{L}a, \mathcal{L}b, \mathcal{L}p, \mathcal{L}q\}$. □

For convenience, from now on, we identify a program $\mathcal{P}(P)$ with its corresponding autoepistemic theory $\mathcal{P}(P, \mathcal{L}P)$ if there is no confusion.

Suppose $F(P)$ is a logic theory, then $F(\mathcal{L}P)$ is the theory obtained from $F(P)$ by replacing each predicate p by $\mathcal{L}p$.

In [48], Yuan characterized both the stable and the well-founded semantics with autoepistemic circumscription theories.

Theorem 5.4

1. *The stable semantics of program $\mathcal{P}(P, \mathcal{L}P)$ is characterized by*

$$ST(\mathcal{P}) \equiv CIR\ (\mathcal{P}(P, \mathcal{L}P); P) \wedge (P \equiv \mathcal{L}P)$$

2. *The well-founded semantics of program $\mathcal{P}$ is characterized by*

$$WF(\mathcal{P}) \equiv CIR\ (\mathcal{P}(P, \mathcal{L}P); P) \wedge CIR\ (\mathcal{P}(\mathcal{L}P, P); \mathcal{L}P)$$

□

By Theorem 5.4, both the stable and the well-founded semantics are now defined in terms of circumscription. In both definitions, the predicates are partitioned into Q and P, with $\mathcal{L}P \in Q$ and others in P in 1), and $\mathcal{L}P \in P$ and others in Q in 2). There is no Z involved.

5.2 ST-resolution: A Proof Procedure for Stable Semantics

In this section, we develop ST-resolution based on the CP-resolution proof-theoretic framework. ST-resolution can extract answers in logic programs with negation with respect to the stable semantics. Like CIR-resolution, ST-resolution is a ME-resolution augmented with CP-resolution nonmonotonic operator. Therefore, it is sound, and is complete under the weakly bounded term size property. For simplicity, we assume the program concerned has at least one stable model.

Definition 5.3 Let

1. $\mathcal{P}(P, \mathcal{L}P)$ be the given program and $GF(\mathcal{P})$, its generalized full program;
2. $\mathcal{P}_{cp}$ be the blocking formulas of $\mathcal{P}$;
3. $\mathcal{P}_{P \equiv \mathcal{L}P}$ be the set of rules which contain $\{p \leftarrow \mathcal{L}p;\ \ p \rightarrow \mathcal{L}p\}$ for each $p \in P$ and $\mathcal{L}p \in \mathcal{L}P$;
4. G be the top goal.

Then, *ST-derivation* of a goal G_n from $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P \equiv \mathcal{L}P} \cup \{G\}$ is a sequence $G = G_0, G_1, \ldots, G_n$ of goals, such that

1. for $1 < i \leq n$, G_i is either
   (a) an extension goal of G_{i-1} and some rule C_i in $GF(\mathcal{P}) \cup \mathcal{P}_{P \equiv \mathcal{L}P}$, followed by a tightening operation; or
   (b) a deduction goal of G_{i-1}; or
   (c) a factorization of G_{i-1}; or
   (d) a completion goal of G_{i-1} and a CP-constructed rule C_i from $\mathcal{P}_{cp}$ or its variant C_i'; and
2. perform a merging, followed by a truncation (ignore E); and
3. the equality formula of G_i is satisfiable;
4. G_i is acceptable.

A *ST-refutation* of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P \equiv \mathcal{L}P} \cup \{G\}$ is a ST-derivation of a pseudo-empty clause of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P \equiv \mathcal{L}P} \cup \{G\}$. □

Compare Definition 5.3 with Definition 4.1, we can see that the only difference between ST-resolution and CIR-resolution is that ST-resolution takes rules from $\mathcal{P}_{P\equiv\mathcal{L}P}$ for its extension operation. This is understandable because by Theorem 5.4, stable semantics of the program $\mathcal{P}(P, \mathcal{L}P)$ is defined as the circumscription of $\mathcal{P}(P, \mathcal{L}P)$ and the formula $P \equiv \mathcal{L}P$.

We will discuss the simplification of ST-resolution in Section B.2, where we will see that the variant C_i' in 1(d) of Definition 5.3 is actually not necessary.

Theorem 5.5 (Soundness of ST-resolution) *Let ´ a ST-refutation of* $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P\equiv\mathcal{L}P} \cup \{\leftarrow \Phi\}$ *end up with a pseudo-empty goal* $\leftarrow E$. *Then, the following holds:*

$$ST(\mathcal{P}) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} S_g(E) \rightarrow \Phi),$$

where $\boldsymbol{y}$ *is a tuple of variables in* E *disjoint to* $\boldsymbol{x}$, *and* $\boldsymbol{z}$ *is the tuple of ex-variables introduced in S-generalization.* □

Proof: By Theorem 5.4, Theorem 4.1 and Definition 5.3. □

Example 5.2 Let $\mathcal{P}$ contain the following rules (also shown are variants):

	The rules	The variants
C_1:	$a \leftarrow \neg\mathcal{L}b$;	
C_2:	$b \leftarrow \neg\mathcal{L}a$;	
C_3:	$c \leftarrow a, \neg\mathcal{L}d$;	
C_4:	$c \leftarrow b$;	$\neg b \leftarrow \neg c$

Let the top goal be $\leftarrow c$.

$\mathcal{P}_{cp}$ contains the following relevant rule, which coincides with the CP-constructed rule from $\mathcal{P}_{cp}$:

C_5: $\neg d$;

$\mathcal{P}_{P\equiv\mathcal{L}P}$ contains the following relevant rules:

C_6: $\neg\mathcal{L}b \leftarrow \neg b$;
C_7: $\neg\mathcal{L}d \leftarrow \neg d$;

The following is a ST-refutation of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P\equiv\mathcal{L}P} \cup \{\leftarrow c\}$:

G_0: $\leftarrow c;$ extension with C_3
G_1: $\leftarrow [c], a, \neg\mathcal{L}d;$ extension with C_7
G_2: $\leftarrow [c], a, [\neg\mathcal{L}d], \neg d;$ extension with C_5
G_3: $\leftarrow [c], a,;$ extension with C_1
G_4: $\leftarrow [c], [a], \neg\mathcal{L}b;$ extension with C_6
G_5: $\leftarrow [c], [a], [\neg\mathcal{L}b], \neg b;$ extension with the variant of C_4
G_6: $\leftarrow [c], [a], [\neg\mathcal{L}b], [\neg b], \neg c;$ reduction
G_7" $\leftarrow \square.$

The empty clause indicates that c is in the stable models. $\square$

The following example is taken from [14], but we don't ground instantiate the program, in order that we can extract answers from $\mathcal{P}$ using ST-resolution. This example also appears in many other research papers where p_1 stands for *move*, and p_2 stands for *winning*.

Example 5.3 Let $\mathcal{P}$ contain the following rules:

C_1: $p_1(x,y) \leftarrow e(\langle x,y\rangle, \langle 1,2\rangle);$
C_2: $p_2(x) \leftarrow p_1(x,y), \neg\mathcal{L}p_2(y);$
C_2': $p_2(x) \leftarrow p_1(x, f(x_0)), \neg\mathcal{L}p_2(f(x_0))$ local Skolemization of C_2;

where f is a Skolem function symbol.

$\mathcal{P}_{cp}$ contains the following rules:

C_3: $\neg p_2(x) \leftarrow \neg p_1(x, f(x_0));$
C_4: $\neg p_2(x) \leftarrow \mathcal{L}p_2(f(x_0));$
C_5: $\neg p_1(x,y) \leftarrow \neg e(\langle x,y\rangle, \langle 1,2\rangle);$

and $\mathcal{P}_{P\equiv\mathcal{L}P}$ contains the following relevant rule:

C_6 $\neg\mathcal{L}p_2(x) \leftarrow \neg p_2(x);$

The following is a ST-derivation of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P\equiv\mathcal{L}P} \cup \{\leftarrow p_2(x)\}$:

G_0: $\leftarrow p_2(x);$ extension with C_2
G_1: $\leftarrow [p_2(x)], p_1(x,y), \neg\mathcal{L}p_2(y);$ extension with C_6
G_2: $\leftarrow [p_2(x)], p_1(x,y), [\neg\mathcal{L}p_2(y)], \neg p_2(y);$

We need a CP-constructed rule here. Following is a CP-refutation of $\mathcal{P}_{cp} \cup \{\leftarrow \neg p_2(y)\}$:

G_0'; $\leftarrow \neg p_2(x_0)$; extension with C_3
G_1': $\leftarrow [\neg p_2(x_0)], \neg p_1(x_0, f(x_0))$; extension with C_5;
G_2'; $\leftarrow \neg e(\langle x_0, f(x_0)\rangle, \langle 1, 2\rangle)$;

The CP-constructed rule is:

C_7: $\neg p_2(x_0) \leftarrow \neg e(\langle x_0, f(x_0)\rangle, \langle 1, 2\rangle)$;

Now we carry on the ST-derivation from G_2 by applying a completion rule with C_7:

G_3: $\leftarrow [p_2(x)], p_1(x, y), \neg e(\langle y, f(y)\rangle, \langle 1, 2\rangle)$; extension with C_1
G_4: $\leftarrow e(\langle x, y\rangle, \langle 1, 2\rangle), \neg e(\langle y, f(y)\rangle, \langle 1, 2\rangle)$;

The computed answer is $\forall x \exists y \forall z_{f(y)}(e(x, 1) \wedge e(y, 2) \wedge (\neg e(y, 1) \vee \neg e(z_{f(y)}, 2))$; since $\neg e(y, 1) \vee \neg e(z_{f(y)}, 2)$ is subsumed by $e(y, 2)$, and $\exists y\, e(y, 2)$ is true, the computed answer is simplified to $\forall x\, e(x, 1)$, which is the same as the expected answer in [14]. □

The following example uses the same program as provided in Example 5.3, but this time the query is $\neg p_2(x)$.

Example 5.4 Let contain the rules same as in Example 5.3. The ST-derivation of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P \equiv \mathcal{L}P} \cup \{\leftarrow \neg p_2(x)\}$ is as follows:

G_0: $\leftarrow \neg p_2(x_0)$;
We need a CP-constructed rule here, which is C_7 in Example 5.3. ST-derivation can be carried on from G_0 by a completion operation with C_7, and we have the following pseudo-empty goal immediately:

G_2: $\leftarrow \neg e(\langle x_0, f(x_0)\rangle, \langle 1, 2\rangle)$;

The computed answer is $\forall x, z_{f(x)}(\neg e(x, 1) \vee \neg e(z_{f(x)}, 2))$; since when $e(x, 1)$ holds, there must exists some $z_{f(1)}$ such that $e(z_{f(1)}, 2)$ is true, the computed answer can be simplified as $\forall x\, \neg e(x, 1)$. This result is exactly the expected in [14] for the query of $\neg p_2(x)$. □

Theorem 5.6 shows that ST-resolution is complete under the same condition as CP-resolution and CIR-resolution, i.e., the weakly bounded term size property.

Theorem 5.6 (Completeness of ST-resolution) *Let $\mathcal{P}(P, \mathcal{L}P)$ be a program possessing the weakly bounded term size property, and $G = \leftarrow \Psi$ be a goal. Suppose we*

have all the ST-refutations of $GF(\mathcal{P}) \cup \mathcal{P}_{cp} \cup \mathcal{P}_{P \equiv \mathcal{L}P} \cup \{\leftarrow \Psi\}$, *which end up with pseudo-empty clauses* $\leftarrow E_1, \ldots, \leftarrow E_i, \ldots$, *then the following holds*

$$ST(\mathcal{P}) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i) \leftrightarrow \Psi),$$

where each $S_g(E_i)$ *is an S-generalization of* E_i *with tuple of variables* $\boldsymbol{z}$ *introduced;* $\boldsymbol{x}$ *is a tuple of variables in* Ψ, *and* $\boldsymbol{y}$ *is a tuple of variables in* $E_1, \ldots, E_i, \ldots$ *disjoint to* $\boldsymbol{x}$. □

Proof: By Theorem 5.4(1), Theorem 4.3 and Definition 5.3. □

5.3 WF-resolution: A Proof Procedure for Well-founded Semantics

In this section, we develop WF-resolution based on the CP-resolution proof-theoretic framework. WF-resolution can extract answers in logic programs with negation with respect to the well-founded semantics. Like CIR-resolution and ST-resolution, WF-resolution is a ME-resolution augmented with CP-resolution. Therefore, it is sound, and is complete under the weakly bounded term size property.

Definition 5.4 Let

1. $\mathcal{P}(P, \mathcal{L}P)$ be the given program and $GF(\mathcal{P})$, its generalized full program;
2. $\mathcal{P}(\mathcal{L}P, P)$ and $GF(\mathcal{P}(\mathcal{L}P, P))$ be denoted by $\mathcal{P}'$ and $GF(\mathcal{P}')$, respectively;
3. $\mathcal{P}_{cp}$ and $\mathcal{P}'_{cp}$ be the blocking formulas of $\mathcal{P}$ and $\mathcal{P}'$, respectively;
4. G be the top goal.

Then, *WF-derivation* of a goal G_n from $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{G\}$ is a sequence $G = G_0, G_1, \ldots, G_n$ of goals, such that

1. for $1 < i \leq n$, G_i is either
   (a) an extension goal of G_{i-1} and some rule C_i in $GF(\mathcal{P}) \cup GF(\mathcal{P}')$, followed by a tightening operation; or
   (b) a deduction goal of G_{i-1}; or
   (c) a factorization of G_{i-1}; or
   (d) a completion goal of G_{i-1} and a CP-constructed rule C_i from either $\mathcal{P}_{cp}$ or $\mathcal{P}'_{cp}$, or its variant C'_i; and

2. perform a merging, followed by an truncation (ignore E); and
3. the equality formula of G_i is satisfiable;
4. G_i is acceptable.

A *WF-refutation* of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{G\}$ is a WF-derivation of a pseudo-empty clause of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{G\}$. □

Compare Definition 5.4 with Definition 4.1 and Definition 5.3, we can see that the difference lies in that WF-resolution take rules from $GF(\mathcal{P}) \cup GF(\mathcal{P}')$ for its extension operation, and the completion associates with a CP-constructed rule from either $\mathcal{P}_{cp}$ or $\mathcal{P}'_{cp}$. This is understandable because by Theorem 5.4, well-founded semantics of the program $\mathcal{P}(P, \mathcal{L}P)$ is defined as the union of $CIR\ (\mathcal{P}(P, \mathcal{L}P); P)$ and $CIR\ (\mathcal{P}(\mathcal{L}P, P); \mathcal{L}P)$;

Theorem 5.7 (Soundness of WF-resolution) *Let a WF-refutation of* $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow \Phi\}$ *end up with a pseudo-empty goal* $\leftarrow E$. *Then the following holds:*

$$WF(\mathcal{P}) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} S_g(E) \rightarrow \Phi),$$

where $\boldsymbol{y}$ *is a tuple of variables in* E *disjoint to* $\boldsymbol{x}$, *and* $\boldsymbol{z}$ *is the tuple of ex-variables introduced in S-generalization.* □

Proof: By Theorem 5.4(2), Theorem 4.1 and Definition 5.4. □

The following Example 5.5 is so called "*Yale shootout*" example which appears in [12], and is originally due to Hanks and McDermott [19]. In the example, we use p_1 to denote *noise*, p_2 to denote *loaded*, p_3 to denote *shoots*, and use p_4 to denote *succ*.

Example 5.5 (Example 7.1 in [12]) Let $\mathcal{P}(P, \mathcal{L}P)$ contain the following rules:

C_1: $p_1(1) \leftarrow p_2(1), p_3(1)$;
C_2: $p_2(2) \leftarrow p_4(0,1), p_2(0), \neg \mathcal{L}p_3(0)$;
C_3: $p_4(0,1)$;
C_4: $p_2(0)$;
C_5: $p_3(1)$;

$\mathcal{P}'$ is obtained from $\mathcal{P}$ by replacing p_i by $\mathcal{L}p_i$, and $\mathcal{L}p_j$ by p_j simultaneously;

Refer to [12], we know that $p_1(1)$ is not in every minimal modes of $\mathcal{P}$, but is in $WF(\mathcal{P})$. Let us see how WF-derivation of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow p_1(1)\}$ can derive $p_1(1)$.

G_0: $\leftarrow p_1(1);$ extension with C_1
G_1: $\leftarrow [p_1(1)], p_2(1), p_3(1);$ extension with C_5
G_2: $\leftarrow [p_1(1)], p_2(1);$ extension with C_2
G_3: $\leftarrow [p_1(1)], [p_2(1)], p_4(0,1), p_2(0), \neg\mathcal{L}p_3(0);$

To this point we need a completion operation. Since $\mathcal{L}p_3(0)$ does not appear in $\mathcal{P}'$, therefore the following rule is in $\mathcal{P}'_{cp}$:

C_6: $\neg\mathcal{L}p_3(0) \leftarrow TRUE;$

The CP-refutation of $\mathcal{P}'_{cp} \cup \{\leftarrow \mathcal{L}p_3(0)\}$ is:

G'_0: $\leftarrow \neg\mathcal{L}p_3(0);$ extension with C_6;
G'_1: $\leftarrow TRUE;$

Therefore the CP-constructed rule is $\neg\mathcal{L}p_3(0) \leftarrow TRUE$, which is used to carried on the WF-derivation left at G_3:

G_4: $\leftarrow [p_1(1)], [p_2(1)], p_4(0,1), p_2(0);$ extension with C_4;
G_5: $\leftarrow [p_1(1)], [p_2(1)], p_4(0,1);$ extension with C_3;
G_6: $\leftarrow \Box;$

Therefore, $WF(\mathcal{P}) \models p_1(1)$ holds. $\Box$

In Example 5.3 and 5.4, the program $\mathcal{P}$ has a unique stable model, which according to Corollary 5.3 coincides with the well-founded model. We thus expect whatever we derived from $\mathcal{P}$ using ST-resolution can also be derived by using WF-resolution.

Example 5.6 Let $\mathcal{P}$ contains the following rules (same as in Example 5.3):

C_1: $p_1(x,y) \leftarrow e(\langle x,y\rangle, \langle 1,2\rangle);$
C_2: $p_2(x) \leftarrow p_1(x,y), \neg\mathcal{L}p_2(y);$
C'_2: $p_2(x) \leftarrow p_1(x, f(x_0)), \neg\mathcal{L}p_2(f(x_0));$ local Skolemization of C_2.

where f is a Skolem function symbol. Let $\mathcal{P}'$ contain the rules C_{1*} and C_{2*}, which are obtained from C_1 and C_2 by replacing p_1 and p_2 by $\mathcal{L}P_1$ and $\mathcal{L}p_2$, while replacing $\mathcal{L}p_2$ by p_2. The Skolemized general form C_{2*} is:

C'_{2*}: $\mathcal{L}p_2(x) \leftarrow \mathcal{L}p_1(x, g(x_1)), \neg p_2(g(x_1));$

where g is a Skolem function symbol different from f.

$\mathcal{P}_{cp}$ contains the following rule:

C_3: $\neg p_2(x) \leftarrow \neg p_1(x, f(x_0))$;
C_4: $\neg p_2(x) \leftarrow \mathcal{L}p_2(f(x_0))$;
C_5: $\neg p_1(x, y) \leftarrow \neg e(\langle x, y\rangle, \langle 1, 2\rangle)$;

Let $\mathcal{P}'_{cp}$ contain the rules $C_{3\star}$, $C_{4\star}$ and $C_{5\star}$, which are obtained from C_3, C_4 and C_5 by replacing p_1 and p_2 by $\mathcal{L}P_1$ and $\mathcal{L}p_2$, $\mathcal{L}p_2$ by p_2, and f by g. The following is a WF-derivation of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow p_2(x)\}$ with query $p_2(x)$:

G_0: $\leftarrow p_2(x)$; extension with C_2
G_1: $\leftarrow [p_2(x)], p_1(x, y), \neg\mathcal{L}p_2(y)$;

We need a CP-constructed rule here. Following is a CP-refutation of $\mathcal{P}'_{cp} \cup \{\leftarrow \neg\mathcal{L}p_2(y)\}$:

G'_0; $\leftarrow \neg\mathcal{L}p_2(x_1)$; extension with $C_{3\star}$
G'_1: $\leftarrow [\neg\mathcal{L}p_2(x_1)], \neg\mathcal{L}p_1(x_1, g(x_1))$; extension with $C_{5\star}$;
G'_2; $\leftarrow \neg e(\langle x_1, g(x_1)\rangle, \langle 1, 2\rangle)$;

The CP-constructed rule is:

C_6: $\neg\mathcal{L}p_2(x_1) \leftarrow \neg e(\langle x_1, g(x_1)\rangle, \langle 1, 2\rangle)$;

Now we carry on the WF-derivation from G_1 with a completion with C_6:

G_2: $\leftarrow [p_2(x)], p_1(x, y), \neg e(\langle y, g(y)\rangle, \langle 1, 2\rangle)$; extension with C_1
G_3: $\leftarrow e(\langle x, y\rangle, \langle 1, 2\rangle), \neg e(\langle y, g(y)\rangle, \langle 1, 2\rangle)$;

The computed answer is $\forall x \exists y \forall z_{g(y)} (e(x, 1) \wedge e(y, 2) \wedge (\neg e(y, 1) \vee \neg e(z_{g(y)}, 2))$. Since $\neg e(y, 1) \vee \neg e(z_{g(y)}, 2)$ is subsumed by $e(y, 2)$, and $\exists y\, e(y, 2)$ is true, the computed answer is simplified to $\forall x\, e(x, 1)$, which is exactly the same as Example 5.3. But this time we use WF-resolution instead of ST-resolution. □

If we change the query to $\neg p_2(x)$, then the WF-refutation of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow \neg p_2(x)\}$ will be the exactly the same as Example 5.4.

Now we show that an object could be in $ST(\mathcal{P})$, but is not in $WF(\mathcal{P})$. Although by Theorem 5.2, a stable model is an extension of the partial well-founded model, there still could be some literals which are in every stable model but not in the well-founded partial model. We will use the same program and the same query as in Example 5.2 to illustrate this point.

Example 5.7 Let $\mathcal{P}$ contain the following rules (same as in Example 5.2):

	The rules	The variants
C_1:	$a \leftarrow \neg \mathcal{L}b;$	$\mathcal{L}b \leftarrow \neg a;$
C_2:	$b \leftarrow \neg \mathcal{L}a;$	$\mathcal{L}a \leftarrow \neg b;$
C_3:	$c \leftarrow a, \neg \mathcal{L}d;$	
C_4:	$c \leftarrow b;$	$\neg b \leftarrow \neg c;$

Let $\mathcal{P}'$ contain the rules $C_{1\star}$ through $C_{4\star}$ which are obtained from C_1 through C_4 by replacing p by $\mathcal{L}p$ and $\mathcal{L}q$ by q, where p and q are atoms.

$\mathcal{P}_{cp}$ contains the following relevant rule, which coincidents with the rule CP-constructed from $\mathcal{P}_{cp}$:

C_5: $\neg d;$

$\mathcal{P}'_{cp}$ contains the following rules:

$C_{5\star}$: $\neg \mathcal{L}c \leftarrow \neg \mathcal{L}a, \neg \mathcal{L}b;$
$C_{6\star}$: $\neg \mathcal{L}c \leftarrow d, \neg \mathcal{L}b;$
$C_{7\star}$: $\neg \mathcal{L}d;$
$C_{8\star}$: $\neg \mathcal{L}a \leftarrow b;$
$C_{9\star}$: $\neg \mathcal{L}b \leftarrow a;$

where $C_{7\star}$ through $C_{9\star}$ coincide with the rules which can be CP-constructed from $\mathcal{P}'_{cp}$.

To clearly demonstrate the WF-derivations, we proceed with the following CP-refutation first, to construct the relevant CP-constructed rules, although in the real situation, such rule should be CP-constructed dynamically whenever needed.

CP-refutation 1:

$\leftarrow \neg \mathcal{L}c;$	extension with $C_{5\star}$
$\leftarrow [\neg \mathcal{L}c], \neg \mathcal{L}a, \neg \mathcal{L}b;$	extansion with $C_{9\star}$
$\leftarrow [\neg \mathcal{L}c], \neg \mathcal{L}a, a;$	extansion with $C_{8\star}$
$\leftarrow b, a;$	

We therefore have CP-constructed rule:

$C_{10\star}$: $\neg \mathcal{L}c \leftarrow b, a;$

CP-refutation 2:

$\leftarrow \neg\mathcal{L}c$; extension with C_{6*}

$\leftarrow [\neg\mathcal{L}c], d, \neg\mathcal{L}b$; extansion with C_{9*}

$\leftarrow d, a$;

We therefore have CP-constructed rule:

C_{11*}: $\neg\mathcal{L}c \leftarrow d, a$;

Now let us start the WF-derivation of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow c\}$.

WF-derivation 1:

G_0: $\leftarrow c$; extension with C_3;

G_1: $\leftarrow [c], a, \neg\mathcal{L}d$; completion with C_{7*}

G_2: $\leftarrow [c], a$; extension with C_1 or variant of C_{2*}

G_3: $\leftarrow [c], [a], \neg\mathcal{L}b$; completion with C_{9*}

G_4: $\leftarrow [c], [a], [\neg\mathcal{L}b], a$.

This is a subsumed goal because it is not acceptable.

WF-derivation 2 can be carried on from G_3 of WF-derivation 1 by extension with variant of C_{4*}

G_4: $\leftarrow [c], [a], [\neg\mathcal{L}b], \neg\mathcal{L}c$; completion with C_{10*}

G_5: $\leftarrow [c], [a], [\neg\mathcal{L}b], [\neg\mathcal{L}c], b, a$.

Again it is a subsumed goal.

WF-derivation 3 can be carried on from G_4 of WF-derivation 2 by completion with variant of C_{11*}

G_5: $\leftarrow [c], [a], [\neg\mathcal{L}b], [\neg\mathcal{L}c], d, a$.

It is a subsumed goal too.

WF-derivation 4 can be carried on from G_0 of WF-derivation 1 by extension with variant of C_4

G_1: $\leftarrow [c], b$; extension with C_2 or variant of C_{1*}

G_2: $\leftarrow [c], [b], \neg\mathcal{L}a$; extension with C_{8*}

G_3: $\leftarrow [c], [b], [\neg\mathcal{L}a], b$.

It is subsumed and WF-derivation 5 can be carried on from G_2 of WF-derivation 4 by extension with variant of C_{3*}

G_3: $\leftarrow [c], [b], [\neg\mathcal{L}a], \neg\mathcal{L}c, \neg d$; extension with C_5

G_4: $\leftarrow [c], [b], [\neg\mathcal{L}a], \neg\mathcal{L}c$; extension with C_{10*}

G_5: $\leftarrow [c], [b], [\neg\mathcal{L}a], [\neg\mathcal{L}c]$, b, a.

Again a subsumed goal. WF-derivation 6 can be carried on from G_4 of WF-derivation 5 by extension with of C_{11*}

G_5: $\leftarrow [c], [b], [\neg\mathcal{L}a], [\neg\mathcal{L}c], d, a;$ extension with C_1 or variant of C_{2*}
G_6: $\leftarrow [c], [b], [\neg\mathcal{L}a], [\neg\mathcal{L}c], d, [a], \neg\mathcal{L}b;$ extension with variant of C_{4*}
G_7: $\leftarrow [c], [b], [\neg\mathcal{L}a], [\neg\mathcal{L}c], d, [a], [\neg\mathcal{L}b], \neg\mathcal{L}c.$
It is subsumed and WF-derivation 7 can be carried on from G_6 of WF-derivation 6 by extension with of C_{9*}
G_7: $\leftarrow [c], [b], [\neg\mathcal{L}a], [\neg\mathcal{L}c], d, [a], [\neg\mathcal{L}b], a.$
Once again it is a subsumed goal.

At this point we have exhausted all the possible WF-derivations of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow c\}$ and all of them are failed, therefore the following holds:

$$WF(\mathcal{P}) \not\models c.$$

□

Theorem 5.8 shows that WF-resolution is complete under the same condition as CP-resolution, CIR-resolution and ST-resolution, i.e., the weakly bounded term size property.

Theorem 5.8 (Completeness of WF-resolution) *Let $\mathcal{P}(P, \mathcal{L}P)$ be a program possessing the weakly bounded term size property, and $G = \leftarrow \Psi$ be a goal. Suppose we have all the WF-refutations of $GF(\mathcal{P}) \cup GF(\mathcal{P}') \cup \mathcal{P}_{cp} \cup \mathcal{P}'_{cp} \cup \{\leftarrow \Psi\}$, which end up with pseudo-empty clauses $\leftarrow E_1, \ldots, \leftarrow E_i, \ldots$, then the following holds*

$$WF(\mathcal{P}) \models \forall \boldsymbol{x}(\exists \boldsymbol{y} \forall \boldsymbol{z} \bigvee_i S_g(E_i) \leftrightarrow \Psi),$$

where each $S_g(E_i)$ is an S-generalization of E_i with tuple of variables $\boldsymbol{z}$ introduced, $\boldsymbol{x}$ is a tuple of variables in Ψ, and $\boldsymbol{y}$ is a tuple of variables in $E_1, \ldots, E_i, \ldots$ disjoint to $\boldsymbol{x}$. □

Proof: By Theorem 5.4, Theorem 4.3 and Definition 5.4. □

Chapter 6

Conclusion

In this dissertation we have discussed various limitations for nonmonotonic proof procedures, which are floundering, unnecessary incompleteness, lack of answer substitutions in negation and inability to extract partial answers when the completion condition is not met. We have also discussed the source of these limitations.

We developed a proof-theoretic framework, CP-resolution, which overcomes the discussed limitations. The following is a list of the major contributions we have made in the dissertation:

1. We established a procedural definition for circumscription based on MIME-resolution.

2. We defined the proof-theoretic framework CP-resolution which is orthogonal to the MIME-resolution. CP-resolution overcomes the limitations with the other nonmonotonic proof procedures. It is proved to be sound with respect to the semantics of circumscription for a single negative literal in $neg(P)$. And more significantly, it is complete if the given set of clauses possesses the weakly bounded term size property, the weakest known completion condition for any nonmonotonic proof procedure. CP-resolution is a one-staged proof-procedure, and can extract partial answers even when the completion condition is not met. It is a proof-theoretic framework in the sense that other one-staged nonmonotonic proof procedures can be developed based on it, and retain its advantages.

3. We developed CIR-resolution based on CP-resolution. To our best knowledge, CIR-resolution is the only proof procedure which can extract answers in circumscription. Other proof procedures have difficulty to handle the semidecidability of first-order logic, and are thus basically for ground cases, giving

only YES or NO answers. CIR-resolution is complete under the condition of the weakly bounded term size property. It is a one-staged proof procedure, and thus can extract partial answers when the completion condition is not met.

4. Based on Yuan's work [48] where two of the best known semantics for logic programming with negation, stable and well-founded semantics, are redefined in terms of circumscription of autoepistemic theory, we further developed ST-resolution and WF-resolution which can extract answers in logic programs with respect to stable and well-founded semantics. Both ST-resolution and WF-resolution are shown to be complete under the weakly bounded term size property. They are one-staged proof procedures and thus can extract partial answers when the completion condition is not met.

The possible future works include:

1. To study and improve the complexity of CP-resolution.
2. To reduce the search space of each resolution.
3. To develop proof procedures for other logic programming and deductive database semantics such as preference semantics.
4. To simplify the computed answers, which are in the form of an equality.

We expect that as a nonmonotonic proof-theoretic framework, CP-resolution will be used to develop one-staged proof procedures for other nonmonotonic reasoning formalisms, semantics of logic programs and deductive databases with negation. Same as the proof procedures we developed in this dissertation, the new proof procedures will enjoy the weaker completion condition, namely the weakly bounded term size property. The will be able to extract answers even when the completion condition is not met due to their one-stagedness.

Bibliography

[1] C.R. Baral and V.S. Subrahmanian. Stable and extension class theory for logic programs and default logics. *Journal of Automated Reasoning*, 8:345–366, 1992.

[2] G. Boolos and R. Jeffrey. *Computability and Logic.* Cambridge University Press, second edition, 1980.

[3] G. Bossu and P. Siegel. Nonmonotonic reasonings and logic programming. *Artificial Intelligence*, 25:13–63, 1985.

[4] D. Chan. Constructive negation based on the completed database. In R. A. Kowalski and K. A. Bowen, editors, *Proc. 5th Symposium/Conference on Logic Programming*, pages 111–125. MIT Press, 1988.

[5] D. Chan. An extension of constructive negation and its application in coroutining. In E. Lust and R. Overbeek, editors, *Proc. North American Conference on Logic Programming*, pages 477–496. MIT Press, 1989.

[6] C. Chang and H. Keisler. *Model Theory.* North-Holland, 1988.

[7] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[8] K. Clark. Negation as failure. In M. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 311–325. Morgan Kaufmann, Los Altos, CA, 1987.

[9] P. M. Dung. Negation as hypotheses: An abductive foundation for logic programming. In *Proc. of 8th Intern. Conference on Logic Programming*, pages 3–17, 1991.

[10] D. Etherington. *Reasoning with Incomplete Information.* PhD thesis, University of British Columbia, 1986.

[11] D. Etherington, R. Mercer, and R. Reiter. On the adequacy of predicate circumscription for closed-world reasoning. In M. Ginsberg, editor, *Nonmonotonic Reasoning*, pages 174–178. Morgan Kaufman Publisher, Los Atlos, CA, 1985.

[12] A. Van Gelder, K. Ross, and J. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proc. the 7th ACM PODS*, pages 221–230, 1988.

[13] M. Gelfond. On stratified autoepistemic theories. In *Proc. AAAI-87*, pages 207–211, 1987.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Proc. 5th Symposium/Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[15] M. Gelfond, H. Przymusinska, and T. Przymusinski. The extended closed would assumption and its relationship to parallel circumscription. In *Proc. of the 5th ACM PODS*, pages 133–139, 1986.

[16] M. Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.

[17] M. Genesereth and N. Nilsson. *Logic Foundations of Artificial Intelligence.* Morgan Kaufman Publishers, Los Altos, CA, 1987.

[18] M. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[19] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proc. AAAI-86*, pages 328–333, 1986.

[20] K. Inoue and N. Helft. On theorem provers for circumscription. In *Proc. of CSCSI'90*, 1990.

[21] P. Kolaitis and H. Papadimitriou. Some computational aspects of circumscription. In *Proc. AAAI-88*, pages 465–469, 1988.

[22] R. Kowalski and D. Kuchner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-3):227–260, 1971.

[23] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[24] J-L. Lassez, M. Mahar, and K. Marriott. Unification revisit. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufman Publishers, 1988.

[25] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, NY, 1981.

[26] V. Lifschitz. Computing circumscription. In *Proc. IJCAI-85*, pages 121–127, Los Angeles, CA, 1985.

[27] J. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, second edition, 1987.

[28] D. Loveland. A linear format for resolution. In *Proc. IRIA Symp. Automatic Demomstration*, pages 147–162. Springer-Verlag, 1970.

[29] D. Loveland. *Automated Theorem Proving: a Logic Basis.* North-Holland, 1978.

[30] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[31] J. McCarthy. Applications of circumscription to formalizing common-sense reasoning. *Artificial Intelligence*, 28:86–116, 1986.

[32] E. Mendelson. *Introduction to Mathematical Logic.* Wadsworth & Brooks, third edition, 1987.

[33] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[34] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[35] H. Przymusinska and T. Przymusinski. Semantic issues in deductive database and logic programming. In A. Banerji, editor, *Sourcebook on the Formal Approaches in Artificial Intelligence.* North Holland, Amsterdam, 1990.

[36] T. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

[37] T. Przymusinski. On constructive negation in logic programming. In *Proc. North American Conference on Logic Programming.* MIT Press, 1989.

[38] T. Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2), 1989.

[39] A. Rabinov. A generalization of collapsible cases of circumscription. *Artificial Intelligence*, 38:111–117, 1989.

[40] R. Reiter. Equality and domain closure in first-order database. *JACM*, 27(2):235–249, 1980.

[41] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[42] R. Reiter. Nonmonotonic reasoning. *Annual Review of Computing Science*, 1987.

[43] R. Reiter. On closed-world data bases. In M. Ginsberg, editor, *Reading in Nonmonotonic Reasoning*. Morgan Haufmann, 1987.

[44] J. Shepherson. Negation in logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufman Publishers, 1988.

[45] A. van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufman Publishers, 1988.

[46] A. van Gelder. The alternating fixpoints of logic programs with negation. In *Proc. of the 8nd ACM Simposium on Principles of Database Systems*, pages 1–10, 1989.

[47] L.Y. Yuan. Autoepistemic logic of first order and its expressive power. *Journal of Automated Reasoning*, 13:69–82, 1994.

[48] L.Y. Yuan. Logic program semantics and circumscription of autoepistemic theories. *Information Processing Letters*, 20:159–164, 1994.

[49] L.Y. Yuan and J.H. You. Autoepistemic circumscription and logic programming. *Journal of Automated Reasoning*, 10:143–160, 1994.

[50] L.Y. Yuan, J.H. You, and C.H. Wang. A proof-theoretical framework for nonmomnotonic reasoning and logic programming. In *Proc. ICCI'91, Lecture Notes in Computer Science 497*, pages 699–708, 1991.

Appendix A

Justification of the Way MIME-resolution is Defined

We can see that MIME-resolution is a bridge between the MIME-precondition $Prec(SF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ and the CP-refutation of $\Pi_{cp} \cup \{\leftarrow \neg p(\boldsymbol{t})\}$. The ground MIME-resolution is defined based on the ground WME-resolution, while MIME-resolution is a *lift* to the general case. As far as MIME-precondition is concerned, MIME-resolution can be defined using any of many other refinements of linear resolution, such as OL-resolution, SL-resolution, and ME-resolution [7, 22, 29]. It only requires that the resolution be sound and complete, and the ancestor resolution can be restricted to a reduction operation like ME-reduction. In term of search space, WME-resolution might be disadvantageous, since it performs no factorization, and tolerates some kind of tautology as well. However, as Loveland put it in [29]

> "...factorization is more trouble than it is worth since one can do without it."

This observation is pertinent in our implementation of CP-resolution. The use of the general form of input chain is almost based on the same reason. We investigate it in detail below.

A.1 Why Introduce Ground WME-resolution?

The original definition of WME-resolution does not demand that the input chains be in general form [29]. The difference between ME-resolution and WME-resolution is that the latter does not perform factorization, does not merge the identical unframed literals, and tolerate a pair of complementary literals unless they are in the same cell.

While the former tries to eliminate the case with a pair of identical of complementary unframed literals in any instance of a goal.

CP-resolution is a process handling a blocking formula from a cut in the tree. In other words, a goal in a CP-derivation has only the knowledge of one unframed literal in the current node, together with all the framed literals which represent the selected literals in its ancestor nodes. A goal in a CP-derivation doesn't know if the only unframed literal from the current node, which it has knowledge, is identical or complementary to another unframed literal in the same node. Therefore, no corresponding operation can ever be taken in CP-derivation even if such a pair does exist. The pair of identical or complementary literals in the same node and the same cell must be introduced from the same input chain at the same step in MIME-resolution. We can eliminate this from happening by fully factorizing and strengthening of each input chain, which we assume are already done.

CP-resolution is orthogonal to MIME-derivation. MIME-resolution should be defined in a way in favor of the implementation of CP-resolution. We are therefore forced to sacrifice search space and build our MIME-resolution on WME-resolution, where only the relationship between an unframed literal in the last cell and a framed literal, or among the framed literals, is needed to be concerned. MIME-resolution pretends that it has no knowledge of any pair of unframed literals in a goal, unless they are in the same cell. Actually, MIME-resolution can be built on weak version of OL-resolution or SL-resolution, should they be defined.

A.2 Why in General Form?

Suppose the MIME-resolution does not require that the input chains be in general form. Suppose there are altogether m MIME-derivations of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$ leading to the MIME-leaves, which are sequences of goals:

$$G_0^i = \leftarrow p(\boldsymbol{t}), \ldots, G_j^i = \leftarrow W_j^i, \ldots, G_{n_i}^i = \leftarrow W_{n_i}^i;$$

associated with the sequence of *mgu's* $\theta_1^i, \ldots, \theta_j^i, \ldots, \theta_{n_i}^i$, for $0 < j \leq n_i$ and $0 < i \leq m$. Let $\theta_i = \theta_1^i \cdots \theta_{n_i}^i$ be composition of *mgu's*. Generally, $\boldsymbol{t} \neq \boldsymbol{t}\theta_i$ for $1 \leq i \leq m$, hence, $Prec(SF(\Pi), \{\leftarrow p(\boldsymbol{t})\})$ should have been defined as:

$$Prec(SF(\Pi), \{\leftarrow p(\boldsymbol{t})\}) = \bigvee_{i=1}^{m} (W_{n_i}^i \wedge e(\boldsymbol{x}, \boldsymbol{x}\theta_i)),$$

where $\boldsymbol{x}$ is tuple of variables appearing in $\boldsymbol{t}$. We can see equalities representing the composition of *mgu's* is involved in its definition.

Due to the nature of CP-resolution, we don't wait for the MIME-derivation to terminate, but blocks the unframed literals like L in $neg(P) \cup any(Q)$ from certain earlier goal W_j^i instead. Note the literal L might not be in the MIME-leaf $W_{n_i}^i$, but its instance $L\theta_{j+1}^i \cdots \theta_{n_i}^i$ is, should we not take the general form. However, we can not wait for the composition of *mgu's* to be generated, in other words, we have no way to obtain $e(\boldsymbol{x}, \boldsymbol{x}\theta_i)$. Something is therefore missing in the CP-goals.

Notice, that

$$\text{CET} \wedge e(\boldsymbol{x}, \boldsymbol{x}\theta_i) \models \forall \boldsymbol{x}(L \leftrightarrow L\theta_{j+1}^i \cdots \theta_{n_i}^i)$$

is true for $1 \leq j < n_i$. It is equivalent if the literal L would be in $W_{n_i}^i$ together with the equality $e(\boldsymbol{x}, \boldsymbol{x}\theta_i)$.

To keep consistency, we add the equality $e(\boldsymbol{x}, \boldsymbol{x}\theta_1^i \cdots \theta_j^i)$ to W_j^i. The equalities are considered in $neg(P) \cup any(Q)$, therefore they would be inherited by the descendent goals. And it can be proven that

$$e(\boldsymbol{x}, \boldsymbol{x}\theta_1^i \cdots \theta_j^i) \equiv e(\boldsymbol{x}, \boldsymbol{x}\theta_1^i) \wedge e(\boldsymbol{x}\theta_1^i, \boldsymbol{x}\theta_1^i\theta_2^i) \wedge \cdots \wedge e(\boldsymbol{x}\theta_1^i \cdots \theta_{j-1}^i, \boldsymbol{x}\theta_1^i \cdots \theta_{j-1}^i\theta_j^i).$$

Therefore, what should be in W_j^i is actually

$$e(\boldsymbol{x}, \boldsymbol{x}\theta_1^i) \wedge e(\boldsymbol{x}\theta_1^i, \boldsymbol{x}\theta_1^i\theta_2^i) \wedge \cdots \wedge e(\boldsymbol{x}\theta_1^i \cdots \theta_{j-1}^i, \boldsymbol{x}\theta_1^i \cdots \theta_{j-1}^i\theta_j^i).$$

This is exactly equivalent to the result of MIME-resolution which demands that the input chains be in general form.

Appendix B

Are Variant Chains Really Needed?

Let Π be a set of clauses. To achieve the completeness of the deduction by using classical resolutions like ME-resolution, OL-resolution and/or SL-resolution, each literal in each clause has to be resolved with [29, 7, 22], while the order of the rest of the literals in the clause could be random. That is why when we take oriented form, we have to first transform Π into $F(\Pi)$ which is the full program of Π, and it is closed under variant chains. We therefore define CIR-resolution in a way that uses $GF(\Pi)$ and the variants of the CP-constructed chains from Π_{cp}. For the same reason, we define SF-resolution and WF-resolution such that they use $GF(\mathcal{P})$, $GF(\mathcal{P}')$, and the variant chains of the CP-constructed chains from $\mathcal{P}_{cp}$ and/or $\mathcal{P}'_{cp}$. We now discuss that in many cases those variant chains might not be needed. Although the discussion does not lead to a weaker condition for the completeness of the resolutions, it does reduce the search spaces.

B.1 Variant Chain Issue with CIR-resolution

First let us consider CIR-resolution. Each chain in $GF(\Pi)$ is certainly needed. While the use of the variant chains of the CP-constructed chains is indispensable, it could be substantially restricted. We give an example to show in certain scenarios, the variant chains are necessary in order to gain completeness.

Example B.1 Let Π contain the clause $\{p \leftarrow q, z;\ \ z \leftarrow \neg q\}$. The only MIME-derivation to compute $Prec(GF(\Pi), \{\leftarrow p\})$ is

G_0: $\quad \leftarrow p;$

G_1: $\leftarrow [p], q, z;$
G_3: $\leftarrow q, \neg q.$

It is a tautology, therefore $Prec(GF(\Pi), \{\leftarrow p\}) \equiv FALSE$, or $\neg p \leftarrow TRUE$ is in Π^*. However, the CP-chains in Π_{cp} are $\{\neg p \leftarrow \neg q;\ \neg p \leftarrow \neg z;\ \neg z \leftarrow q\}$, and the CP-derivation to construct the CP-chains are:

CP-refutation 1:
G_0: $\leftarrow \neg p;$
G_1: $\leftarrow \neg q;$
CP-refutation 2:
G_0: $\leftarrow \neg p;$
G_1: $\leftarrow [\neg p], \neg z;$
G_2: $\leftarrow q.$

The CP-constructed chains in Π^* are $\{\neg p \leftarrow \neg q;\ \neg p \leftarrow q\}$. Although $\neg p$ is a logic consequence of Π^*, there does not exist a chain in Π^* such that it subsumes $\neg q$. Therefore to derive $\neg p$, we need to resolve $\neg p \leftarrow \neg q$ against $\neg p \leftarrow q$, and obviously the variant chain is needed for this sake. □

Recall by Theorem 3.4 $CIR(\Pi; P; Z) \equiv \Pi \cup \Pi^*$, and Π^* contains

$$p \leftrightarrow Prec(GF(\Pi), \{\leftarrow p\})$$

for each $p \in pos(P)$ (note that $(p \leftarrow Prec(GF(\Pi), \{\leftarrow p\})$ is implied by Π). For a model M of $\Pi \cup \Pi^*$, there does not exist a model M' of $\Pi \cup \Pi^*$ such that $M' \leq M \bmod (P, Z)$. Assume the other way around. Let $Prec(GF(\Pi), \{\leftarrow p\})$ consist of $\{\leftarrow W_1, \ldots, \leftarrow W_n\}$. If $M \models p$ and $M' \models \neg p$, then $M \models W_i$ and $M' \not\models W_i$ for $1 \leq i \leq n$. Since $W_i \in neg(P) \cup any(Q)$ by definition, then either M' interprets some $q \in any(Q)$ in a different way as M does, or $M' \models p'$ while $M \models p$ for some $p \in pos(P)$. It contradicts the assumption that $M' \leq M \bmod (P, Z)$.

The intuition behind $\Pi \cup \Pi^*$ is for each subgoal $\neg p$ in the goal, for $p \in pos(P)$, we replace it by $\neg Prec(GF(\Pi), \{\leftarrow p\})$, so that no negative subgoal $\neg p$ can exist eventually. Therefore, if for each conjunction of literals D such that $\Pi^* \models \neg p \leftarrow D$, there exists a conjunction D' of literals such that $\neg p \leftarrow D'$ is a CP-constructed chain in Π^* and $\neg p \leftarrow D'$ subsumes $\neg p \leftarrow D$, then the variant chains would not be needed. Unfortunately this is not the case as shown in Example B.1.

The reason why $\Pi^* \models C$, where C is a chain, and there does not exist a CP-constructed chain C' which subsumes C, is because that CP-resolution does not recognize tautology chain in MIME-derivation, which is orthogonal to it.

Proposition B.1 *If there exists a MIME-tree of* $SF(\Pi) \cup \{\leftarrow p\}$ *which has no tautology leaf node, then for every chain* C *such that* $\neg p \leftarrow \neg Prec(SF(\Pi), \{\leftarrow p\})$ *implies* C*, there must exist a chain* C' *in* $\neg p \leftarrow \neg Prec(GF(\Pi), \{\leftarrow p\})$ *such that* C' *subsumes* C. □

Proof: For $\neg p \leftarrow \neg Prec(GF(\Pi), \{\leftarrow p\})$ to imply $C = \neg p \leftarrow W$, W must be subsumed by a minimal blocking formula W' of the perfect cut of some MIME-tree of $SF(\Pi) \cup \{\leftarrow p\}$. Refer to the proof of Theorem 3.11, any blocking formula of any cut of any MIME-tree can be reached through CP-resolution. If there is no tautology leaf node in some MIME-tree of $SF(\Pi) \cup \{\leftarrow p\}$, then W' is reachable through a CP-refutation of $\Pi_{cp} \cup \{\leftarrow \neg p\}$. □

Each $\neg Prec(SF(\Pi), \{\leftarrow p\})$ is a set of sequences of literals in $pos(P) \cup any(Q)$, the only possible pair of complementary literals must be in $any(Q)$.

By Proposition B.1, a variant of a CP-constructed chain C is needed as an input chain for a completion rule, only if the selected literal is in $any(Q)$ and is introduced into the goal through a completion rule with a CP-constructed chain C' whose head is the same literal as C.

B.2 Variant Chain Issue with ST-resolution

In $\mathcal{P}(P, \mathcal{L}P)$, each $\neg\mathcal{L}p$ is in $neg(Q)$, while each p is in $pos(P)$, and $\neg p$ in $neg(P)$. By the analysis in Section B.1, the variant of a CP-constructed rule is needed only if there exists some tautology leaf node in the MIME-tree which contains a pair of complementary literals in $any(Q)$. However, in each rule of $\mathcal{P}$, there exists only $\neg\mathcal{L}p$, and $\mathcal{L}p$ could thus never appear in the leaf node of any MIME-tree. Therefore, the variant of each CP-constructed rule is not needed to achieve the completion of ST-resolution.

B.3 Variant Chain Issue with WF-resolution

For the same reason as stated in Section B.2, the variant of the CP-constructed rule from both $\mathcal{P}_{cp}$ and $\mathcal{P}'_{cp}$ are not needed.

In [49], Yuan and You defined *well-founded constrain theory* as follows.

Definition B.1 [Definition 5.1 in [49]] Let $\mathcal{P}(P, \mathcal{L}P)$ be an autoepistemic theory and $CIR(\mathcal{P}) = CIR(\mathcal{P}(P, \mathcal{L}P); P)$. Then the *well-founded constraint theory* of $\mathcal{P}$, denoted by $WFT(\mathcal{P})$, is iteratively constructed as follows:

$W^0 = \emptyset$.
$W^{n+1} = \{\mathcal{L}\rho \mid \rho$ *is an objective atom and* $\mathcal{P} \wedge W^n \models \rho\}$
$\cup \{\neg\mathcal{L}\rho \mid \rho$ *is an objective atom and* $CIR\,(\mathcal{P}) \wedge W^n \models \neg\rho\}$

Then $WFT(\mathcal{P})$ is defined as the least fixpoint of sequence W^i. □

Yuan and You further claim in Theorem B.2 that $WFT(\mathcal{P})$ is transformable to be equivalent to the well-founded semantics if $\mathcal{P}$ is a logic program.

Theorem B.2 (Theorem 5.8 in [49]) *Let* Π *be a logic program and* $\Pi'(P, \mathcal{L}P)$ *be its autoepistemic translation. Let* $WFT(\Pi')$ *be the well-founded constraint theory of* $\Pi'(P, \mathcal{L}P)$. *Let* $T = \{\rho | \mathcal{L}\rho \in WFT(\Pi')\}$ *and* $F = \{\rho | \neg\mathcal{L}\rho \in WFT(\Pi')\}$, *then* $\langle T, F\rangle$ *is the (3-valued) well-founded model of* Π, *where* T *is the set of atoms assigned to TRUE and* F *the set of the atoms assigned to FALSE.* □

Now we show that the full program of $\mathcal{P}$ is not needed. Let

$$C = p_0 \leftarrow p_1, \ldots, p_m, \neg\mathcal{L}p_{m+1}, \ldots, \neg\mathcal{L}p_n$$

be a rule in $\mathcal{P}$. For $\neg\mathcal{L}p_0$ to be in W^i, i.e., for

$$CIR\,(\mathcal{P}) \cup W^{i-1} \models \neg p_0$$

to hold, it requires either $\mathcal{L}p_j \in W^{i-1}$, for some $m < j \leq n$, or $CIR\,(\mathcal{P}) \cup W^{i-1} \models \neg p_j$, for some $1 \leq j \leq m$, due to the monotonic increasing of W^i, and the single canonical mode of well-founded semantics. Therefore, the variant of C

$$C' = \neg p_j \leftarrow p_1, \ldots, \neg p_0, \ldots, p_m, \neg\mathcal{L}p_{m+1}, \ldots, \neg\mathcal{L}p_n$$

is either subsumed, or $CIR\,(\mathcal{P}) \cup W^{i-1} \models \neg p_j$, and is thus not needed to derive $\neg p_j$ in W^{i+1}.

Appendix C

Complicated Proofs

Proof of **Theorem 3.2:** Let $G_0 = \leftarrow p(\boldsymbol{t})$ be the top goal of a MIME-derivation of $SF(\Pi) \cup \{G_0\}$, and $G'_0 = \leftarrow \neg p(\boldsymbol{t})$ be the top goal of a CP-derivation of $\Pi_{cp} \cup \{G'_0\}$. It is easy to see that G'_0 is a blocking formula of G_0, which is a trivial cut in MIME-tree of $SF(\Pi) \cup \{\leftarrow p(\boldsymbol{t})\}$.

Assume G'_i is a goal of CP-derivation of $\Pi_{cp} \cup \{G'_0\}$ which is a blocking formula of a cut Ξ_1 in a MIME-tree of $SF(\Pi) \cup \{G_0\}$, i.e., let

$$G_i = \leftarrow [\rho_0], W_1, \ldots, [\rho_j], W_{j+1}, \ldots, [\rho_{s-1}], W_s, \rho_s,$$

be a node in Ξ_1 where ρ_s is selected, and

$$G'_i = \leftarrow [\neg\rho_0], D_1, \ldots, [\neg\rho_j], D_{j+1}, \ldots, [\neg\rho_{s-1}], D_s, \neg\rho_s,$$

be a goal of CP-derivation of $\Pi_{cp} \cup \{G'_0\}$ and $\neg\rho_s$ is selected. G'_i and G_i are orthogonal to each other, and ρ_s is in the selection range for MIME-derivation if and only if $\neg\rho_s$ is in the selection range of CP-derivation. Let

$$\begin{aligned} G_{i+1,1} &= \leftarrow [\rho_0], W_1, \ldots, [\rho_j], W_{j+1}, \ldots, [\rho_{s-1}], W_s, [\rho_s], W_{s+1,1}, \\ &\ldots\ldots \\ G_{i+1,m} &= \leftarrow [\rho_0], W_1, \ldots, [\rho_j], W_{j+1}, \ldots, [\rho_{s-1}], W_s, [\rho_s], W_{s+1,m} \end{aligned}$$

be all the children (before the $(pos(P) \cup any(Z))$-truncation is performed) of G_i in the MIME-tree of $SF(\Pi) \cup \{G_0\}$. Note each $G_{i+1,k}$ could be either a $(pos(P) \cup an(Z))$-extension or a $(neg(P) \cup an(Z))$-reduction, followed by a tightening. We show that G'_{i+1} must be in a form:

$$G'_{i+1} = \leftarrow [\neg\rho_0], D_1, \ldots, [\neg\rho_j], D_{j+1}, \ldots, [\neg\rho_{s-1}], D_s, [\neg\rho_s], \neg d_1, \ldots, \neg d_m,$$

for any possible combination of $\neg d_1, \ldots, \neg d_m$, such that d_k is a literal in $W_{s+1,k}$, $1 \leq k \leq m$. Namely, G'_{i+1} is a blocking formula of a cut Ξ_2 in the MIME-tree of

$SF(\Pi) \cup \{G_0\}$ such that Ξ_2 is obtained from Ξ_1 by replacing G_i with all its children (before the $(pos(P) \cup any(Z))$-truncation is performed). Since changing the ordering among the children of G_i does not affect the finiteness of the MIME-tree, we assume they are ordered corresponding to $\neg d_1, \ldots, \neg d_m$.

The goal G_{tmp} generated through CP-extension is orthogonal to each children of G_i which are yielded through extension inference rule of MIME-derivation because of the orthogonality between the full program and its blocking formulas. For each inequality $\neg e(\boldsymbol{t}_1, \boldsymbol{t}_2)$ introduced through tightening operation in the MIME-derivation, it has an effect as if the chain $\rho_s \leftarrow W_i, \neg e(\boldsymbol{t}_1, \boldsymbol{t}_2)$ is in $SF(\Pi)$, instead of $\rho_s \leftarrow W_i$. Therefore both chains $\neg\rho_s \leftarrow \neg\omega_1, \ldots, \neg\omega_{i-1}, \neg\omega_i, \neg\omega_{i+1}, \ldots, \neg\omega_m$, and $\neg\rho_s \leftarrow \neg\omega_1, \ldots, \neg\omega_{i-1}, e(\boldsymbol{t}_1, \boldsymbol{t}_2), \neg\omega_{i+1}, \ldots, \neg\omega_m$, could be used as input chain in CP-derivation. CP-tightening is a simulation of CP-extension using the latter kind of chain as input chain. Either CP-tightening is performed or not, the goal G_{tmp} is orthogonal to each child of G_i which is yielded through extension inference rule of MIME-derivation.

In MIME-resolution, the reduction inference rule adds an equality $e(\boldsymbol{t}_1, \boldsymbol{t}_2)$, which has an effect that the chain $\rho_s \leftarrow e(\boldsymbol{t}_1, \boldsymbol{t}_2)$ is in $SF(\Pi)$, and is used as an input chain for extension. Therefore the inequality $\neg e(\boldsymbol{t}_1, \boldsymbol{t}_2)$ should be dynamically added to each CP-chain with head $\neg\rho_s$ on this occasion, which is implemented by CP-reduction. As a result, the goal G_{tmp} generated through CP-extension is also orthogonal to each child of G_i which is yielded through reduction inference rule of MIME-derivation.

Truncation does not affect the orthogonality. Therefore G'_{i+1} is orthogonal to each child of G_i. It follows that G'_{i+1} is a blocking formula of Ξ_2 which is one level lower than Ξ_1. This completes the proof of Theorem 3.2. □

Proof of **Lemma 3.3:** Assume that D_i is minimal, i.e., there does not exist D'_i such that $D'_i \subset D_i$ and $\Pi \models p(\boldsymbol{a}) \leftarrow D'_i$.

We will prove the lemma by proving that there exists a MIME-leaf $\leftarrow W_j$ of $F(\Pi) \cup \{\leftarrow p(\boldsymbol{a})\}$ such that $D_i \subseteq W_j$ and $\Pi \cup D_i \models W_j$.

$\Pi \cup D_i \cup \{\neg p(\boldsymbol{a})\}$ is inconsistent. Let Π' be obtained from Π by removing each clause containing literal L_k, for all literal $L_k \in D_i$, and then removing $\neg L_k$ from each clause. Apparently, $\Pi' \cup \{\neg p(\boldsymbol{a})\}$ is inconsistent. There must exist a ME-refutation of $F(\Pi') \cup \{\leftarrow p(\boldsymbol{a})\}$. Now reinstate each literals in $neg(P) \cup any(Q)$ we selected during the ME-refutation to the sequence of goals, i.e., there is a MIME-refutation of $F(\Pi') \cup \{\leftarrow p(\boldsymbol{a})\}$ which ends up with a MIME-leaf $\leftarrow W'$. Apparently $F(\Pi') \cup \{\leftarrow W'\}$ is inconsistent, and we can carry on the ME-derivation of $F(\Pi') \cup \{\leftarrow W'\}$ and reach an empty clause. Now further reinstate the literals in D_i,

and we obtain a goal $\leftarrow D_i, W'$, which is actually a MIME-leaf of $F(\Pi) \cup \{\leftarrow p(a)\}$.

$\Pi \cup D_i \models D_i \wedge W'$ must hold, because we can carry on a ME-derivation of $F(\Pi) \cup \{\leftarrow D_i, W'\}$ and reach the goal $\leftarrow D_i$. This completes the proof of Lemma 3.3. □

Proof of **Lemma 4.2:** Let Π_s and Π_s° be the ground instantiations of Π and Π° to the universe U_s, respectively, which is the underlying universe U augmented with ground Skolem terms. Let Π_L and Π_L° be the ground instantiations of Π and Π° to the underlying universe U, respectively. Let E_S be the second-order equality formula which is a collection of mappings from $\mathsf{U}_s - \mathsf{U}$ to U.

$$E_S^i \cup \Pi_s \cup \Pi_s^\circ \models \Psi \tag{1}$$

holds for an arbitrary E_S^i from $\mathsf{U}_s - \mathsf{U}$ to U, assuming $E_S^i \cup \Pi_s \cup \Pi_s^\circ$ is satisfiable.

For each clause C in Π_s (respectively Π_s°), there is a clause C' in Π_L (respectively Π_L°) such that:

$$E_S^i \models C \leftrightarrow C'. \tag{2}$$

By Formulas (1) and (2), we have:

$$E_S^i \cup \Pi_\mathsf{L} \cup \Pi_\mathsf{L}^\circ \models \Psi. \tag{3}$$

Note, Π_L contains neither equality nor Skolem term. With E_S^i, each ground equality in Π_L° can be explicitly evaluated as TRUE or FALSE with respect to CET.

We delete each clause from Π_L° which contains a TRUE equality literal, and then delete each false equality literal from the remaining clause. The resulting set is denoted as Π_L''. The following formula (without equality formula and E_S^i) must be true:

$$\Pi_\mathsf{L} \cup \Pi_\mathsf{L}'' \models \Psi.$$

By completeness of ME-resolution, there must exist a ME-refutation of $F(\Pi_\mathsf{L} \cup \Pi_\mathsf{L}'') \cup \{\leftarrow \Psi\}$, which ends up with an empty clause.

Now reinstate the deleted ground equality literals back to each chain, and correspondingly, each goal. We obtain ME-refutation of $F(\Pi_\mathsf{L} \cup \Pi_\mathsf{L}^\circ) \cup \{\leftarrow \Psi\}$, which ends up with a pseudo-empty clause $\leftarrow E_i^\star$ such that

$$E_S^i \models E_i^\star. \tag{4}$$

By *Lifting Lemma*, there exists a ME-refutation of $F(\Pi \cup \Pi^\circ) \cup \{\leftarrow \Psi\}$, which ends up with a pseudo-empty clause $\leftarrow E_i$ and there exists a non-Skolem substitution θ_i such that

$$E_i\theta_i = E_i^\star. \tag{5}$$

A CIR-derivation of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\}$ is a ME-derivation of $F(\Pi \cup \Pi^\circ) \cup \{\leftarrow \Psi\}$. Since Π possesses the weakly bounded term size property, by Theorem 3.11, each chain in Π° can be CP-constructed. There must exist a CIR-refutation of $\Pi \cup \Pi_{cp} \cup \{\leftarrow \Psi\}$ which ends up with a pseudo-empty clause $\leftarrow E_i$ and there exists a substitution θ_i such that Formula (5) holds.

The following holds by definition:

$$S_g(E_i\theta_i) = S_g(E_i)\theta_i, \tag{6}$$

for non-Skolem substitution θ_i. By (4), (5) and (6), we have:

$$\bigvee_j E_S^j \models \forall \boldsymbol{z}(\bigvee_i S_g(E_i)\theta_i), \tag{7}$$

the tuple of variables $\boldsymbol{z}$ is universally quantified since no matter how you assign $\boldsymbol{z}$ with the ground terms, it would always correspond to a mapping from ground Skolem terms to ground non-Skolem terms, and satisfies some mapping in $E_S^1, \ldots, E_S^j, \ldots$.

By definition we also have the following formula:

$$\Pi \cup \Pi^\circ \models_S \bigvee_i E_S^i \tag{8}$$

Combine (7) and (8), it follows immediately that:

$$\Pi \cup \Pi^\circ \models_S \forall \boldsymbol{z}(\bigvee_i S_g(E_i)\theta_i). \tag{9}$$

This completes the proof of Lemma 4.2. □